V-II Design Engineering → Covers and shows or holds within Part-1

→ Design Engineering encompasses the set of principles, concepts and practices that leads to the development of a high quality s/w (or) product.

→ Design principles establish an overriding → more important than other consideration philosophy that guides the designer in the work that is performed.

→ Design concepts must be understood before the mechanics of design practice are applied.

→ It leads to the creations of various representations of s/w.

Design:-

→ Design is what virtually → almost every engineer wants to do.

→ It is the place where creativity rules - customer's requirements, business needs, and technical considerations all come together in the formulation of a product (or) s/y.

Important:-

① Design allows a s/w engineer to model the s/y (or) product that is to be built.

② Design is the place where s/w quality is established.

* Design process:-

→ Software design is an iterative process through which requirements are translated into a "blue print" for constructing the s/w.
↳ Design plan (or) technical drawing

→ The design is represented at a high level of abstraction (Quality of dealing with ideas)

→ The design process is an step by step and repetitive procedure where s/w is simulated according the requirements.

→ The simulation (emition of concrete material) will be in detailed (high level of abstraction).

→ Design process is required for better quality software.

→ This quality is achieved by a series of Formal Technical reviews (FTR).

Characteristics (or) Goals

① Design must consider and fulfill all the customer's requirement that are implicit and must implement explicit requirements contained in the analysis model. (It must implement all the explicit requirements contained in the analysis model).

② Design must be readable, understandable and as a guide for those who generate code and for the those whose test & subsequent support of the s/w.

③ Design should provide a complete picture of the s/w, addressing data, functional & behavioral domains.

Design Quality (s/w quality is defined as how well the s/w works)

→ These are the few guidelines which are the characteristics of a good design. → characteristics of items used by designers

Quality guidelines

① A design should exhibit an architecture that,
   a) has been created using recognizable architectural styles (or) patterns.
   b) is composed of components that exhibit good design characteristics
   c) can be implemented in an evolutionary fashion, thereby facilitating implementation and testing. 4.

② A design should be modular (partitioning elements).

③ A design should contain distinct representation of data, architecture, interfaces and components. (To understand all)

④ design should expose the data structures useful for the classes to be implemented. (In design all you will decide which DS to be used).

⑤ design should consist of components that have independent functional characteristics.

⑥ A design should lead to interface that reduce the complexity. (manage to make it easier)

⑦ design should be achive from Iterative method (Checking continuously)

The above are the mandatory guide lines to be followed while designing.

Quality Attributes (FURPS)

→ The FURPS quality attributes represent a target for all S/W design:

Functionality: functionality is assessed by evaluating the feature set and capability of the program, generality of functions and security overall system.

Usability: is assessed by considering human factors, overall aesthetics, consistency and documentation.

Reliability: is evaluated by measuring the frequency and severity of failure, the accuracy of output results, and their mean-time-to-failure (MTTF), the ability to recover from failure, and the predictability of the program.

Performance: is measured by processing speed, response time, resource consumption, throughput, and efficiency.

Supportability: combines the ability to extend the program (extensibility), adaptability, serviceability - these three attributes represent a more common term maintainability.

Design Concepts:- Following is a set of fundamental S/W design concepts

1. Abstraction
   - highest level of abstraction.
   - lowest level of abstraction.
   - procedural abstraction
   - data abstraction.
2. Architecture
   - Structured models.
   - Framework models
   - Dynamic models
   - Process models
   - Functional models.
3. Patterns
4. Modularity
5. Information Hiding.

(6) Functional Independence
- Cohesion
- Coupling

(7) Refinement

(8) Refactoring

(9) Design classes
- User interface classes
- Business domain "
- process classes
- persistent classes
- s/y classes

1) Abstraction :- (hiding unwanted data/code) By hiding the modul will be easily understood.

→ Abstraction simply means to hide the details to reduce complexity and increase efficiency (or) quality.

→ Different levels of abstraction are necessary and must be applied at each stage of design process so errors that are present can removed efficiently.

+ At the highest Level of abstraction :- a solution is stated in broad terms using the language of the problem environment.
(broad terms means we are just giving a high level of details but not in details about that)

+ At lower level of abstraction : A more detailed description of the solution is provided.

+ procedural abstraction : Refers to sequence of instructions that have a specific and limited functions. (step-by-step) Ex:- open for a door.

+ Data Abstraction : It is a named collection of data that describes a data object
Ex:- door type, swing direction, opening mechanism, weight, dimensions.
encompassed Attributes

2) Architecture :- (design architecture) / (most popular architecture is data abstraction architecture (or) data centered architecture / data inheritance all client and s

It is the structure (or) organization of program & program components (modules), their interaction and structure of data that are used by components → used to represent major elements & their interactions.

Architectural models

* Structured models - organized collection of program components
* Framework models - increase the level of design abstraction
* Dynamic models - Represents the behavioral aspects indicating changes as a function of external events.
* Process models - Focus on the design of the business (or) technical process.
* Functional models - Can be used to represent the functional hierarchy of a s/y.

3) pattern :- (a repeated form)

→ The pattern simply means a repeated form of design in which the shape is repeated several times to form a pattern

→ The pattern in the design process means the repeation of a solution to a common occurring problem with a certain context

The intent of each pattern is to provide a discription that enables a designer to determine.

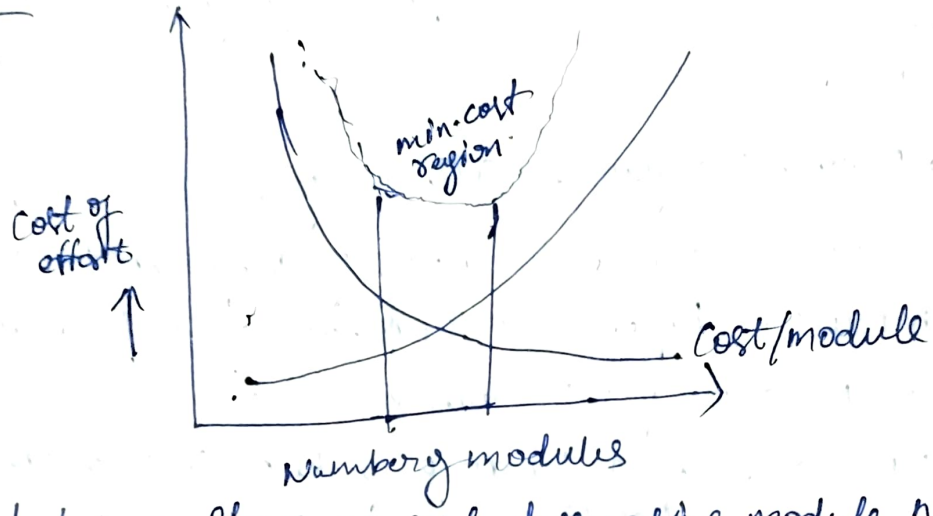
* whether the pattern is capable to the current work.
* whether the pattern can be reused.
* whether the pattern can serve as a guide for developing a similar but functionally (or) structurally different pattern.

4) Modularity :- (sub divided the system)

Modularity simply means to divide the s/w (or) project into smaller parts to reduce the ~~components~~ complexity of the s/w (or) product.

→ In the same way, modularity in design means to subdivided a s/w into smaller parts so that task parts can be created independently & their designing & construction will easily.

Figure:-



→ The left curve shows as cost decreasing module numbers are increasing.
→ The right curve shows as modules increasing the cost is also increasing.

Hence always select medium module based on project cost and complexity.

5) Information Hiding :- (hide the information)

Information hiding simply means to hide the information, the information so that it cannot be accessed by unwanted party (users (or) Hackers).

6) Functional Independent :- (independent methods)

As a project has multiple functions/methods they all should be independent to each other.

Cohesion :- It is an indication of the relative functional strengths of a module. → It is the concept of intra module.
→ Represents the relationship within in two module.

Coupling :- It is an indication of the relative interdependence among modules.
→ It is a concept of inter module.
→ It represents the relationship b/w the modules.

7) Refinement :- (remove impurities)

→ Refinement simply means to find the impurities of present & increase the quality.

→ Refinement is very necessary to find out any bugs/errors of present optimisation is also done here.

→ Refinement is actually a process of elaboration.

8) Refactoring :- (reconstruct something)

→ Refactoring simply means to reconstruct something in such a way that it doesnot affect the behavior of any other features.

→ Refactoring in s/w design means to reconstruct the design to reduce complexity and to make it simple without affecting the behavior (or) its functions.

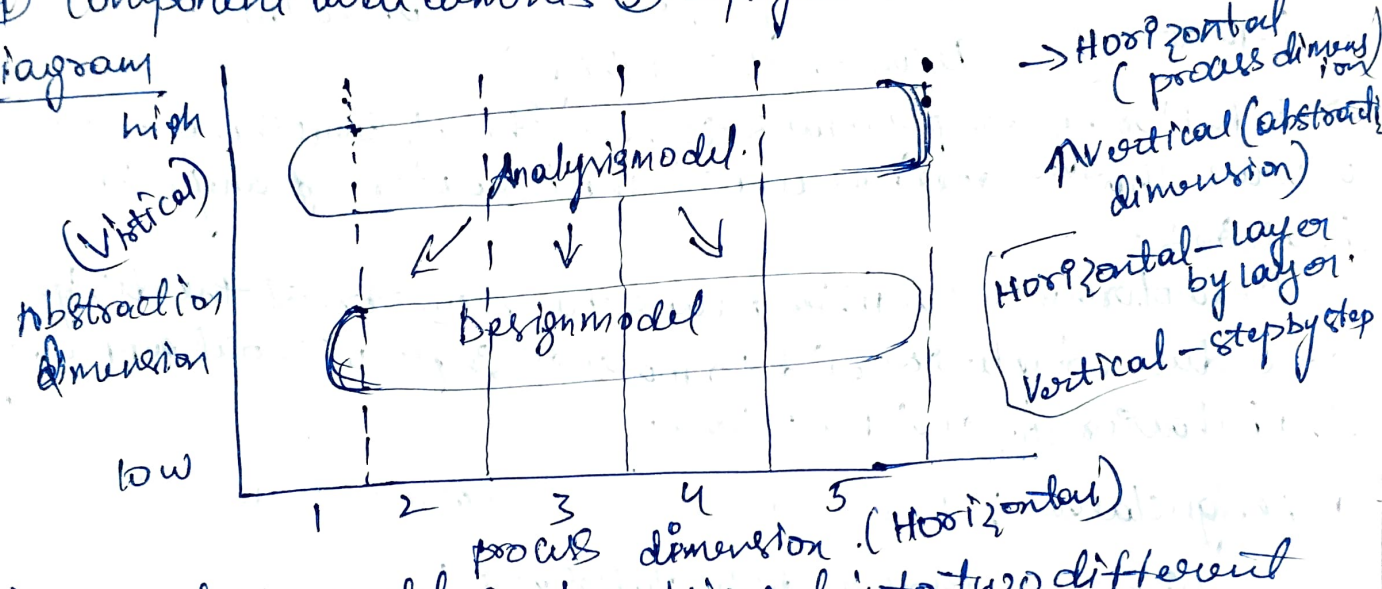
9) Design classes : The model of s/w is defined as a set of design classes.

Different types of design classes are :

* User interface classes : define all abstractions that are necessary for human computer interaction.
* Business Domain :
* process classes :
* persistent classes :
* s/y classes :

Design model:- Designing a model is an important phase and is a multiple process that represents the data structure, program structure, interface characteristic, and procedural details.
→ It is mainly classified into 5 categories (or) elements.
① Data / class elements ② Architectural elements ③ Interface elements
④ Component level elements ⑤ Deployment - level elements.

Diagram



→ The design model can be viewed into two different dimensions.
1) Horizontally - The process dimension indicates the evolution of the parts of the design model each design task executed.
2) Vertically - The abstraction dimension represent the level of details as each element of the analysis model is transferred into design model & then iteratively refined.
→ The elements of the design model use many of the same UML diagram used in the analysis model.
→ They are only refined & elaborated.
→ More implementation specific is provided.

We will elaborating them of the diagram is in 2 steps, we will making into 5/6 steps.

(1) Data design elements :-

→ It is also referred as "data architecturing".
→ Data design creates a model of data and or information that is represented at a high level of abstraction (customer/user view of data)
→ As in any project data plays a key role, hence designing it properly helps a lot in the further (coding, testing)
→ The structure of data is the most important part of the s/w design.

(2) Architectural Design elements :-

→ Architectural design is nothing but a photocopy of the end software.
→ The ADE provides us overall view of the system.
To achive the following
a) we need to have knowledge on the application domain.
b) Relationship b/w UML diagram.
c) we need to have proper architectural styles & patterns

(3) Interface Design Elements :-

→ These tells us how information flow into and out of the system
→ ~~It includes~~ they communicate b/w the components defined as part of architecture.
→ Following are the important elements of the interface Design

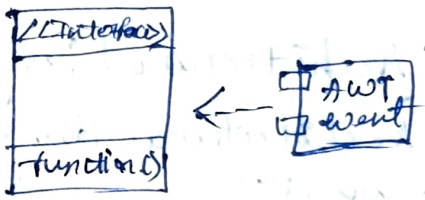
Include a) The user interface.
b) External interface.
c) Internal interface.

(4) Component level design elements :-

→ The component level design for the software completely describes the internal details of each software component.
→ It provides all details of a given ~~statement~~ S/w Component.
→ In order to achive this the component design describes the representation of data structure.

UML Component diagram :-

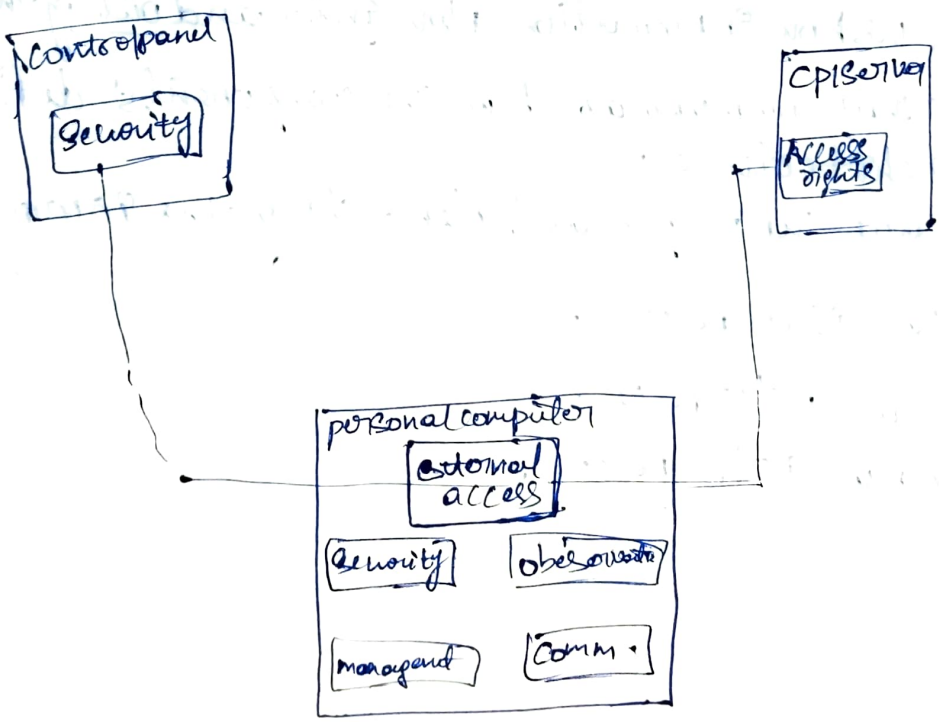
The component is described into the interface (within)



(5) Deployment level design Elements :-

→ These indicate how software functionality & subsystem will be allocated within the physical computing environment that will support the S/w.

fig: Deployment level diagram.



* Creating an architectural design

→ Software Architecture I

→ Data design II

→ Architectural styles and patterns III

→ Architectural design IV

(components are nothing but functions (or) modules used in the program)

→ Software Architecture I → (describing structure of something).

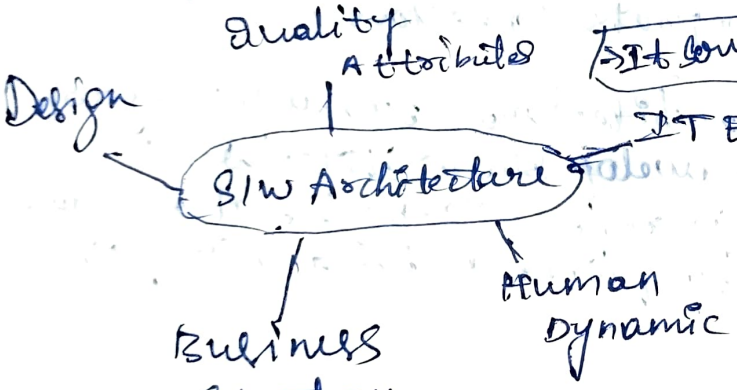
→ S/w architecture of a program (or) computing system is the structures of the system. If we want build (or) develop any architecture we need some structures which compose → S/w components

→ details (about data structures & algorithms which are hidden in the S/w)

→ relationship among components

→ data flow, control flow, dependencies.

5000 LOC (just design) 500 to 5 just write



→ It serves as a blueprint for a S/y.

→ It provides an abstraction to manage the sy complexity & establish a comm. and coordination mechanism among components.

use →

→ It reduce the risk associated with the construction S/w

Imp:-

→ It provides a representation facilitates communication among all stack holders.

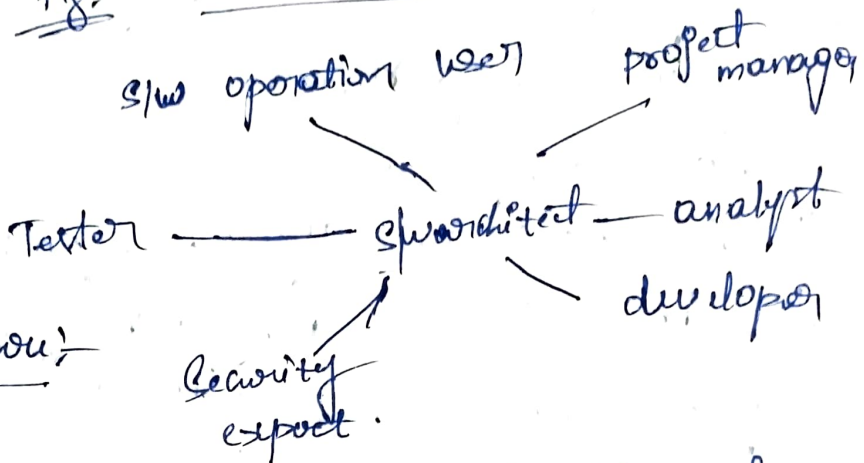
→ It highlights early design decisions.

→ It is relatively small, how the system is structured and how its components work together.

Adv

1) Stakeholder Comm.
2) System analyst
3) Large-scale reuse.

Fig:- S/w architecture



Elements of S/w Architecture:-

① Components:- It is a abstract unit of s/w instruction and internal state which provides transformation of data.

② Connectors:- The connectors are abstract mechanism that communicate, cooperate and coordinate among components.

③ Configuration topologies:- It is nothing but structure of architectural relationships among components, connectors.

④ System models:- Different architectural styles are going to be used to develop s/w architecture.

Architecture:- Architectural design represents the structure of data and program components that required to build a computer based s/y.

II. Data Design:- Data Design actions translate data objects into data structure, further into a database architecture.

(It is a region of storage that contains a value (or) group of values)

-> It contains two levels.

① Data Design at the Architectural level:-

-> In todays s/w business environment, where a lot of data and database are used to the challenge is to extract useful information.

-> To solve this challenge, business, IT community has developed data mining techniques, also called knowledge Discovery in Database (KDD), to extract (or) import ~~info~~ from the database.

→ An alternate solution called as data warehouse is a large, independent db that has access to data that are stored in db.

(2) Data Design at the component level:- Data design at the component level, focuses on representation of data structures that are directly accessed by one (or) more s/w components.

Principles:-

→ A s/w design and programming language should support the specification and realization of abstract data types.
→ A library of useful data structure and operations that may be applied to them should be developed.
→ The representation of a data structure should be known only to those modules that must make direct use of data contained within the structure.
→ All data structure & the operations to be performed on each should be identified.
→ The systematic analysis principles applied to function & behaviours should be applied to data.

III. Architectural Styles

→ The builder has used an architectural style as a descriptive mechanism to differentiate the house from other styles.
→ Each style describes a system category that encompasses
1) A set of components (eg: a database, computational modules) that perform a function required by a system.
2) A set of connectors that enables "communication, coordination & cooperation" among components.
3) Constraints that define how components can be integrated to form the system and
4) Semantic models that enable a designer to understand the overall properties of a system by analysing the known properties of its constituent parts.

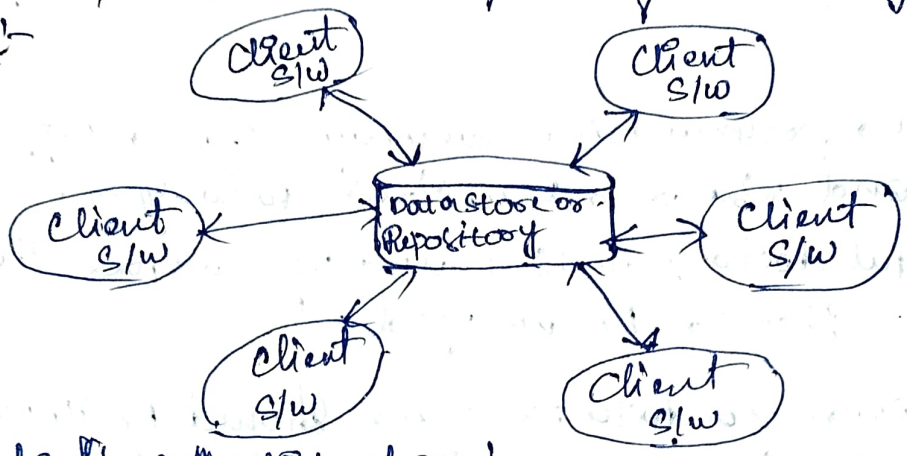
1) Data-Centered Architecture:- A data store (files (or) db) resides at the center of its architecture and is accessed frequently by other components that update, add, delete (or) modify data within the store.

→ Data-centered architecture helps integrity.

→ Pass data b/w clients using the blackboard mechanism.

→ The processes are independently executed by the client components.

Figure:-

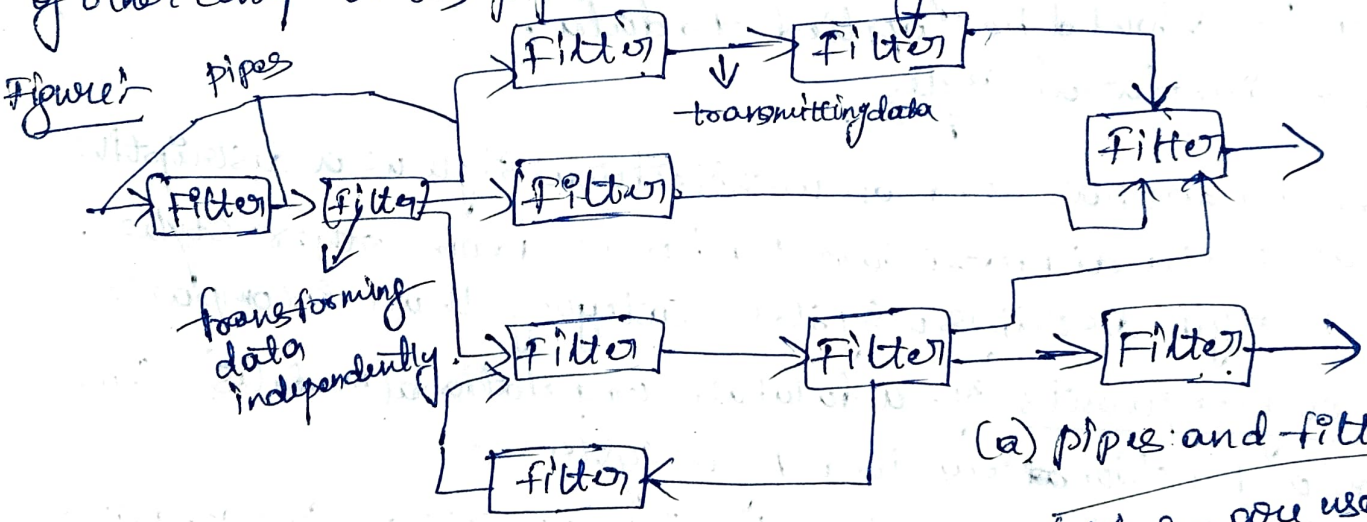


2) Data Flow Architecture:-

→ This architecture is applied when i/p data are to be transformed through a series of computational (or) manipulative components into o/p data.

Eg:- Apipe-and-filter pattern, filters transforming data independently of other components, pipes transmitting data.

Figure:-



(a) pipes and filters.

(b) batch sequential.

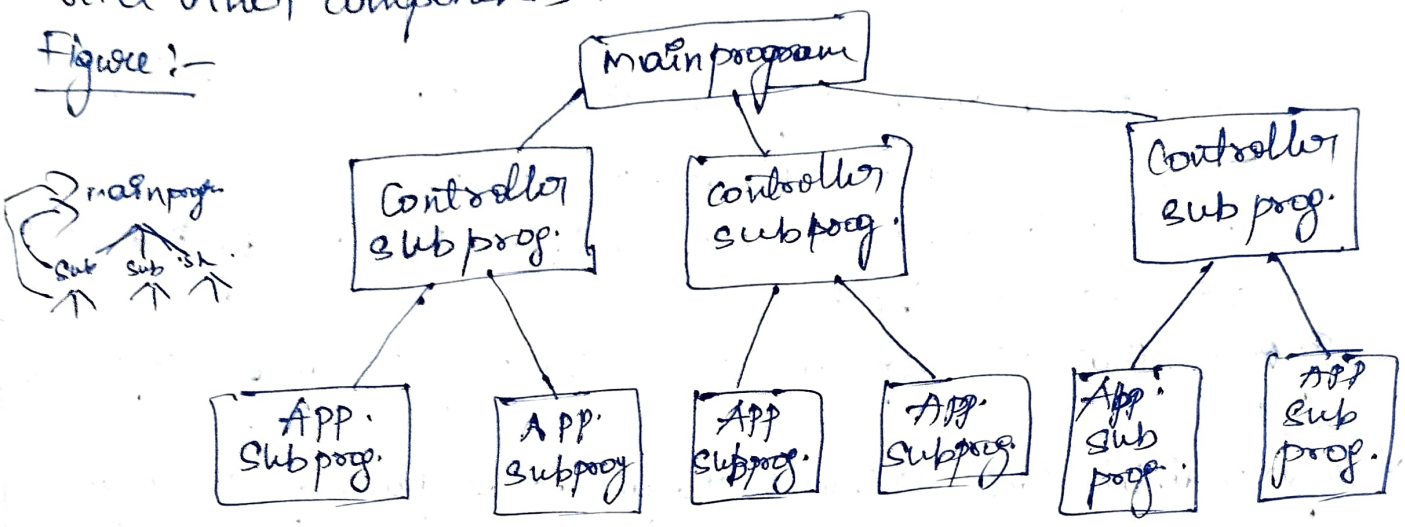
pipes - are used to transmitting data.

filters - are used to transforming data independently

3) Call and Return Architecture

Main program / subprogram Architecture :- Decompose function into a control hierarchy where a "main" program invokes a number of program components, which in turn may invoke still other components.

Figure :-



4) Object-oriented Architecture :- The components of a s/y encapsulate data and the operations that must be applied to manipulate the data. Communication and Coordination b/w components are accomplished via message passing.

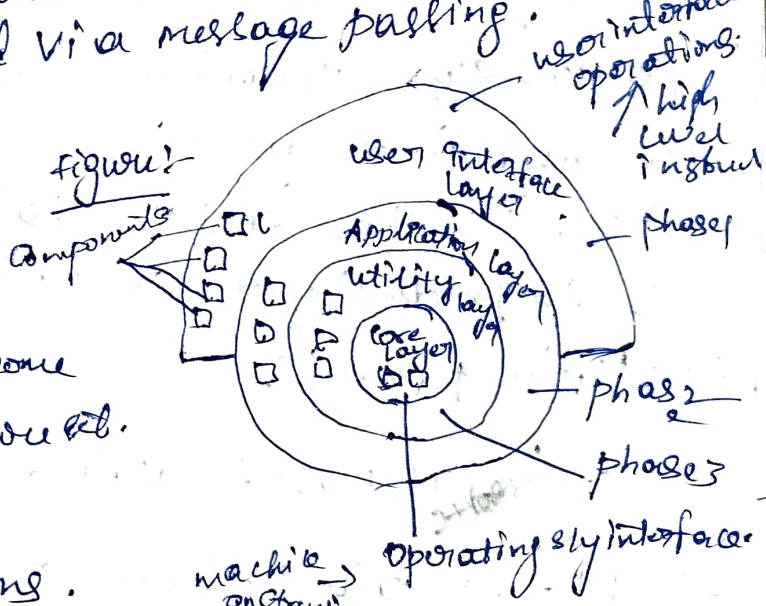
5) Layered Architecture :-

→ A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set.

→ At the outer layer, components service user interface operations.

→ At the inner layer, components perform operating s/y interfacing.

figure :-



IV Architectural Design

→ S/w Architectural Design represents the structure of data and program components that are required to build a Computer based system.

→ The architectural design is not a operational (ready for use) s/w, but it is representation.

→ Architectural design, representing the s/w in Context:
- Sys engineer must model the context.
- Context diagram - models in which the s/y interacts with external entities

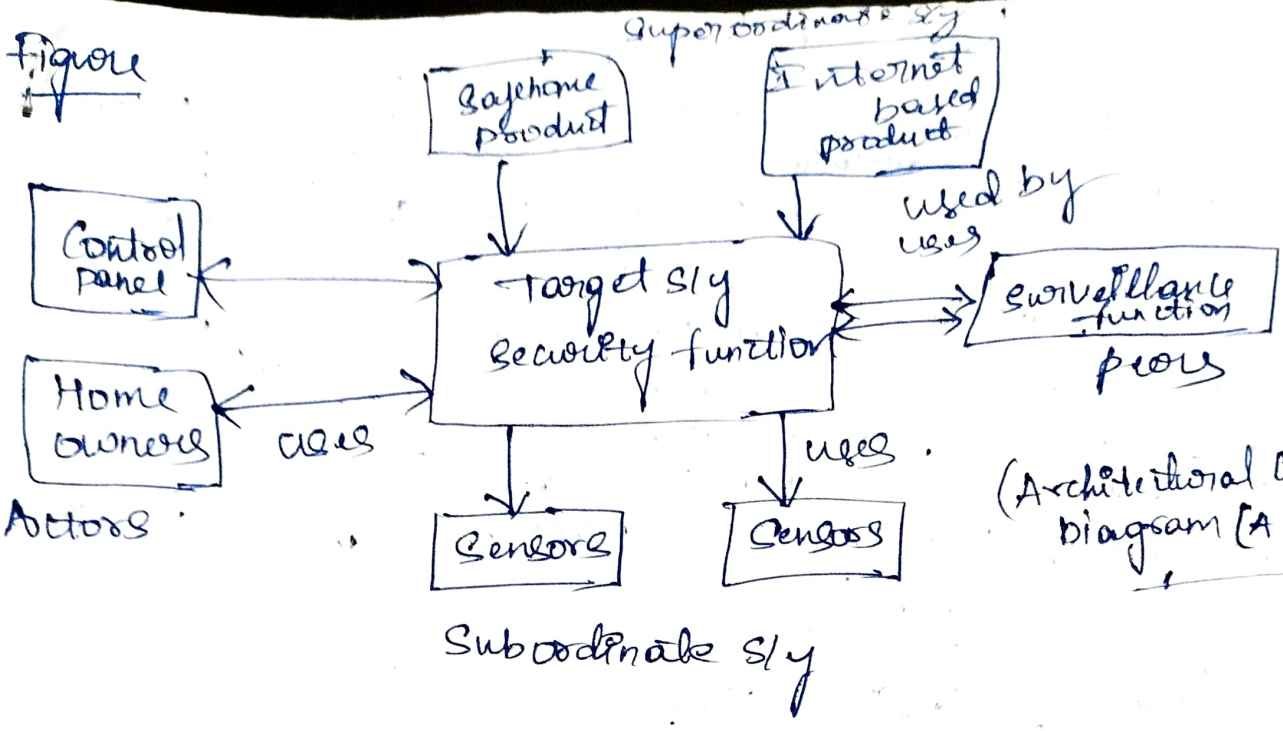
→ System that interoperate with the target s/y are represented as:-

* Superordinate s/y:- using the target s/y as part of some higher level processing scheme.
* Subordinate s/y:- used by the target s/y to provide data (or) processing needed to compute the target s/y.
* Peer level s/y:- producing or consuming information needed by pwr and the target s/y.
* Actors:- those entities that interact with the target s/y by producing (or) consuming information that is necessary for requisite.

→ Interfaces must be defined (relationship)

→ All the data that flow into (or) out of the target s/y must be identified.

Figure



* Architectural patterns:-

→ An architectural pattern is a general, reusable solution to a commonly occuring problem in s/w architecture within a given context.

① Layered pattern:- This pattern can be used to structure programs that can be decomposed into groups of subtasks, each of which is at a particular level of abstraction. Each layer provides services to the next higher layer.

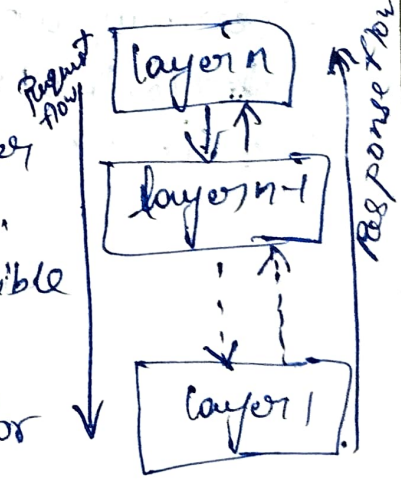
⇒ 4 layers

1) presentation layer:- (User interface layer) the UI layer where we see and enter data into an application.

2) Business layer:- (Domain layer) This layer is responsible for executing business logic as per the request.

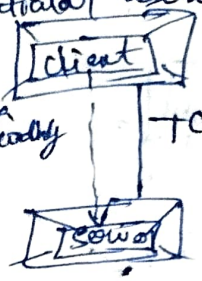
3) Application layer:- (Service layer) This layer acts as a medium for comm. b/w the 'presentation layer' and 'data layer'.

4) Data layer:- (persistance layer) This layer has a db for managing data.



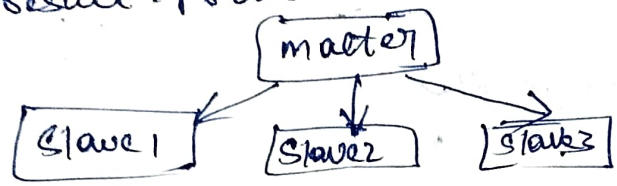
⇒ Open System Interconnection (OSI) model has 7 layers.

(2) Client-Server pattern :- It has two major entities. They are a server and multiple clients. → Here the server has resources (data, files) and client requests the server for particular resource. The server processes the request & responds back accordingly.



Eg:- Email, www, File sharing apps, Banking etc.

(3) Master-Slave pattern :- This pattern consists of two parts i.e master & slaves. The master component distributes the work among identical slave components, and computes a final result from the results which the slaves return.

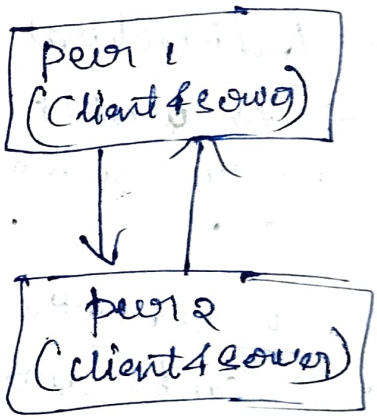


(4) ~~Pipe-fi~~ peer-to-peer pattern :-

→ In this pattern, individual components are known as peers.

→ peers may function both as a client, requesting services from other peers, and as a server, providing services to other peers.

→ A peer may act as a client or as a server (or as both, and it can change its role dynamically with time.



* Modeling Component-level design — Unit-2 — part-B

Component level design :-

→ Component level design occurs after the first iteration of the architectural design.

→ A component level design can be represented using some intermediate representations (eg:- graphical, tabular (or) text based) that can be translated into source code.

→ In design basically, we follow 3 steps (levels) architectural design, analysis design, & component design. So, component level design is the last level. (Stage)

→ Once the component level design stage done then we will go automatically coding stage.

→ The design of data structure, interfaces, and algorithm should conform to well established guidelines to helps us avoid the introduction of errors.

→ If the component level design done 100% correctly the basic errors like structural & algorithm errors will be resolved.

Component :-

→ A component is a modular building block for computer s/w
(or)
→ It is a small module of a huge program (or) project.

→ "A component is a modular, deployable and replaceable part of a system that encapsulate implementation and exposes a set of interfaces".

→ It is an interface b/w two or more components & it is a component it is connected to multiple components also it does a particular functionality.

→ Based on different point of view the meaning of Component changes

* Object oriented view
* Conventional view
* process related view.

→ Component level design is the definition and design of components and modules after the architectural design phase.
→ Component level design defines the data structure, algorithms, interface characteristics, and communication mechanisms allocated to each component for the s/y development.

Views

1) Object-oriented View: Under the object oriented view, an individual component contains a set of collaborating classes, or a group of objects with common properties, operations and relationships to other components. Classes can be divided into two categories.
* Analysis classes (problem domain) representative of people, locations, or other physical objects.
* Design classes (infrastructure) classes with in the context of s/w architecture that allow an application to be executed.

2) Conventional View :- A component is viewed as a functional element of the s/w, meaning that it is part of a program that incorporates the processing logic and required internal data structures to perform its designated tasks.

3) Process Related View :- It especially advocates for the advantage of component reusability, requiring that each process-related component contains:
- A complete description of the interface.
- A definition of the specific function that each component is meant to perform.
- The communication & collaboration with other components that each component requires.

* Characteristics of Components :-
1. Reusability
2. Replaceable
3. Not context specific
4. Extensible
5. Encapsulated.
6. Independent.

* Designing Class based components

→ When an object-oriented s/w engineering approach is chosen, Component level design focuses on the elaboration of problem domain specific classes and the definition and refinement of infrastructure classes contained in the requirement model.

I) Basic design principles

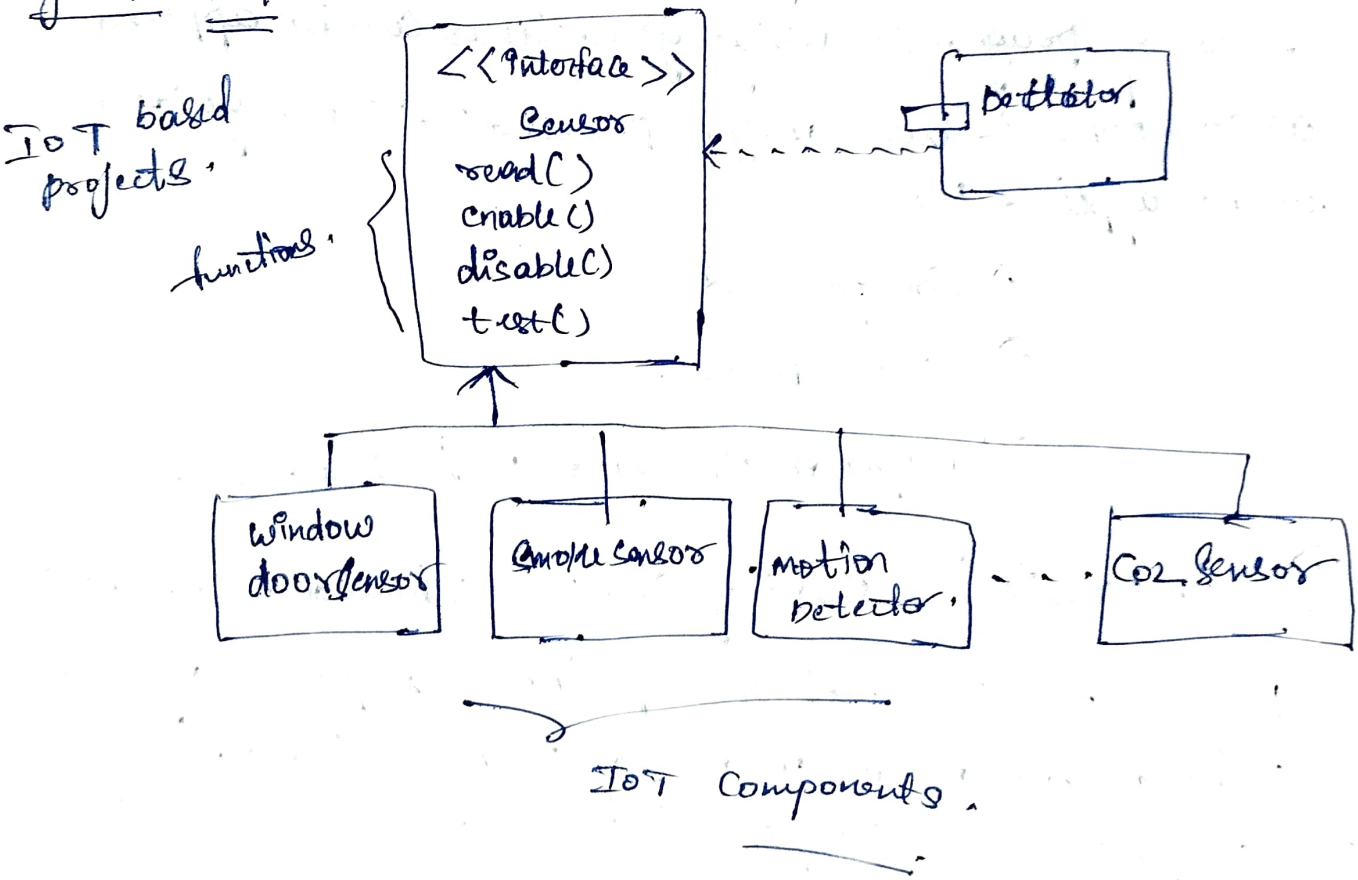
Four basic design principles are applicable to component-level design and have been widely adopted when object-oriented s/w Engineering applied.

1) The open closed principle (ocp):-

→ A module (or) Component should be open for extension but closed for modification.

→ The designer should specify 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.

Figure:- OCP



2) The LISKOV Substitution principle (LSP) ⇒ Replace of main class.

→ Subclasses should be substitutable for their base class.

→ ~~For suppose subclass is there~~ If there is subclass ~~from~~ main class the subclass should represent all the characteristic of main class.

3) Dependency inversion principle (DIP)

→ Depend on abstractions. Donot depend on concretions.

→ The more a component depend on other concrete components (rather than on the interface) the more difficult it will be to extend.

4) The interface segregation (separation) principle (ISP)

→ many client specific interface are better than one general purpose interface.

→ for server class, specialized interfaces should be created to serve major categories of client.

II) Component-Level Design Guidelines

A set of pragmatic design guidelines can be applied as component-level design proceeds. These guidelines can be applied to

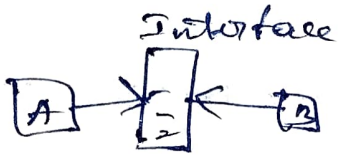
1. Components
2. Interfaces.
3. Dependencies and Inheritance.

① Components :- Naming conventions should be established for components that are specified as part of the architectural model and then refine & elaborated as part of the component level design model.

→ Architectural Component names should be drawn from the problem domain and should have meaning to all stakeholders who view the architectural model.

(5) Interfaces :- Interface provides important information about communication and collaborations b/w two components

figure:-



→ It acts as a broker b/w two components. It just has some functions & operations

(3) Dependencies & Inheritance :-

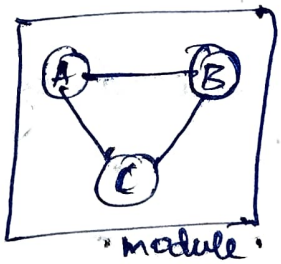
It is a good idea to model dependencies from ~~left~~ to right and inheritance from bottom to top.

III) Cohesion :- (maximum Cohesion) is important

→ Cohesion is a measure of the degree to which the elements (or) components of the module are functionally related.

→ A good s/w design will have high cohesion.

Figure



Types

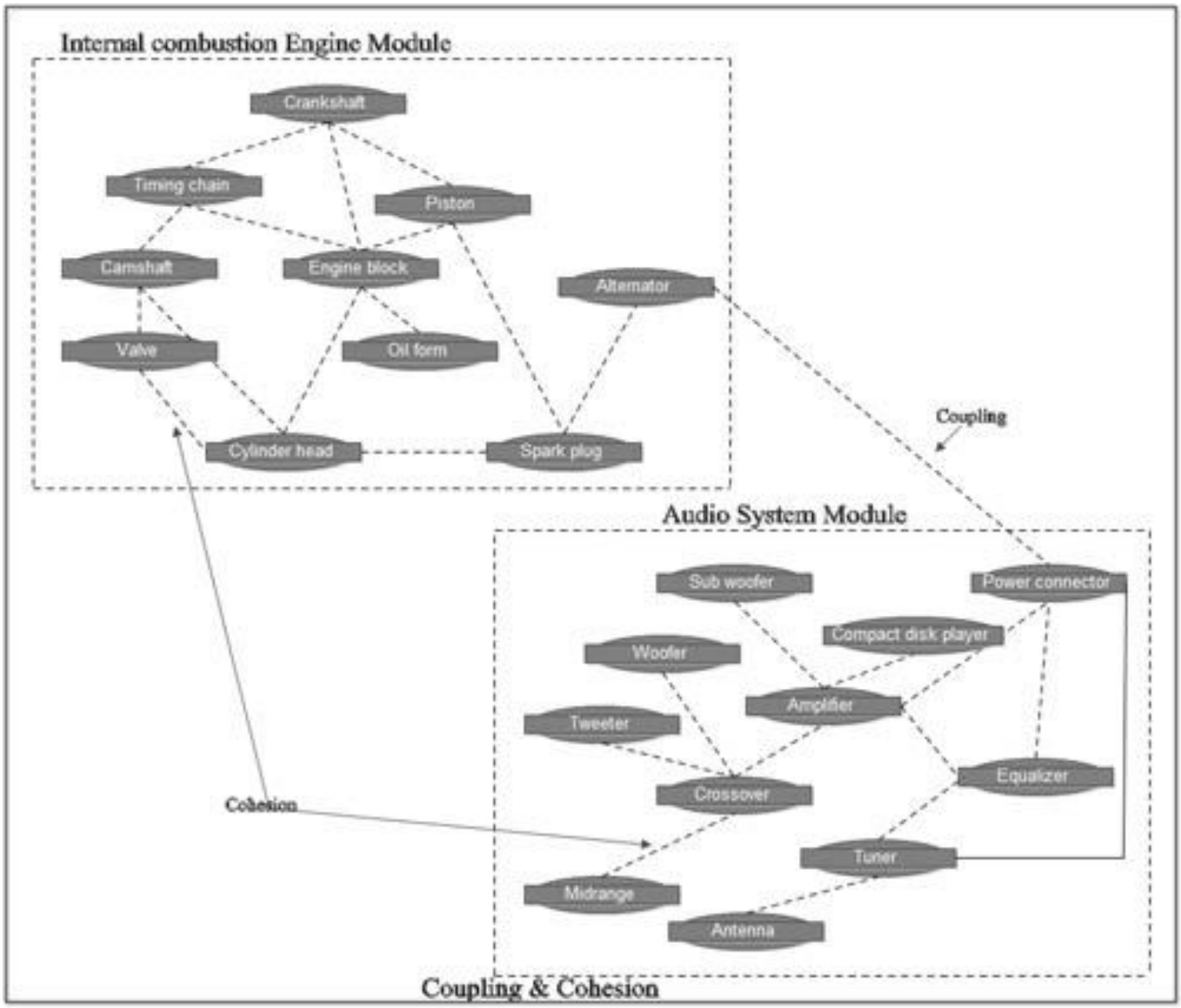
Best

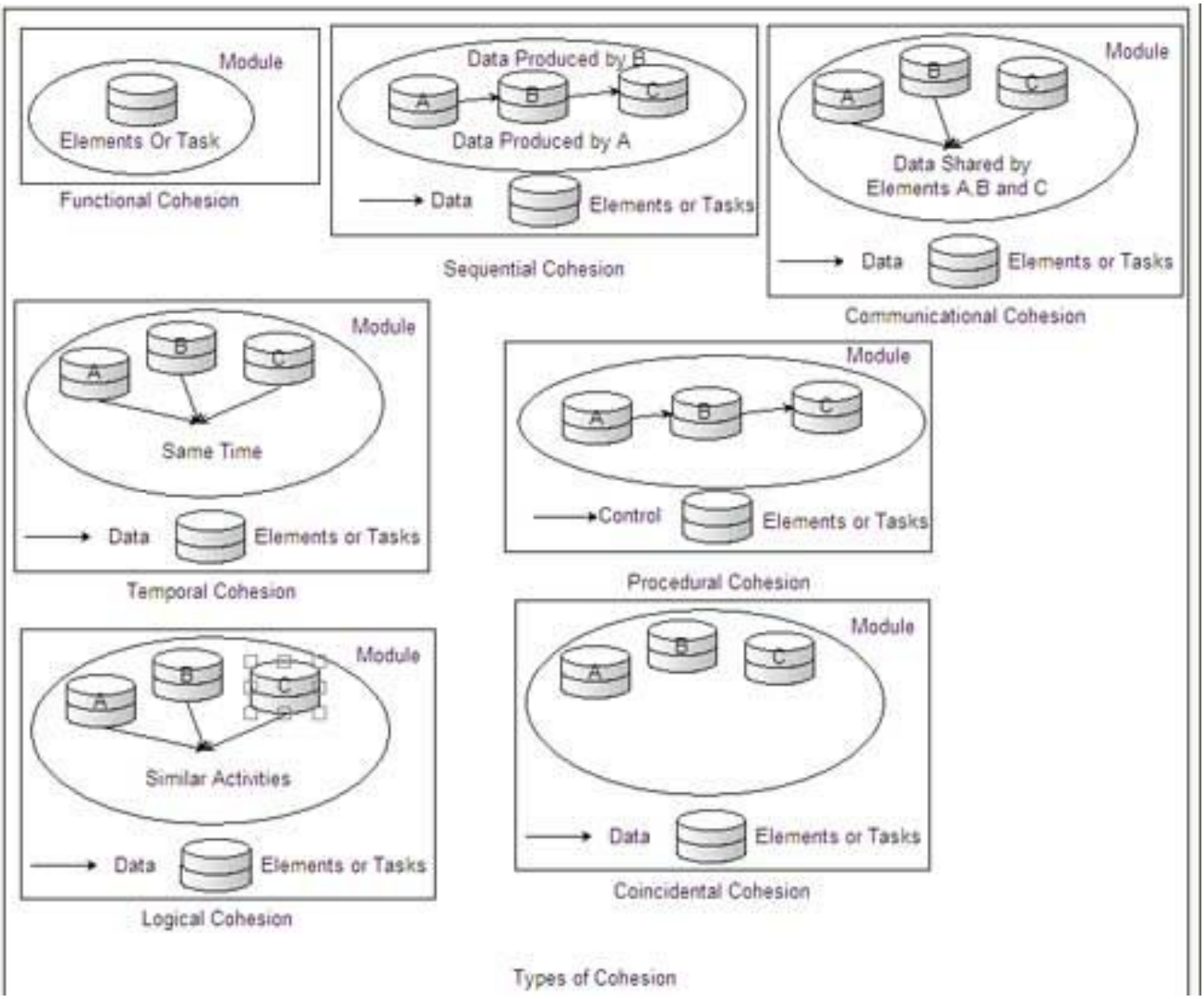
1) Functional Cohesion
2) Sequential "
3) Communication "
4) procedural "
5) Temporal "
6) Logical "
7) Coincidental "

high

low

worst

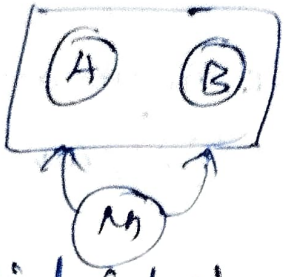

Internal combustion Engine Module
Crankshaft
Timing chain
Piston
Camshaft
Engine block
Alternator
Valve
Oil form
Cylinder head
Spark plug
Coupling
Audio System Module
Sub woofer
Power connector
Compact disk player
Woofer
Amplifier
Tweeter
Equalizer
Crossover
Cohesion
Tuner
Midrange
Antenna
Coupling & Cohesion



Types of Cohesion

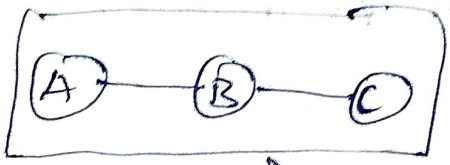
1) Functional Cohesion :- Two (or) more functions performing similar task should be placed in a single module. (If they are related)

Fig:-



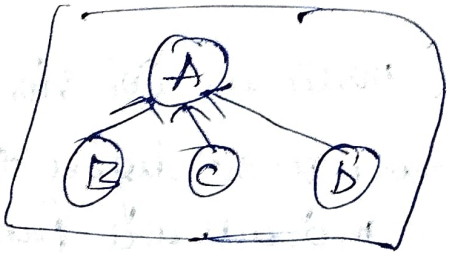
2) Sequential Cohesion :- An elements outputs some data that become the input for other elements data flow between the part.

Fig:-



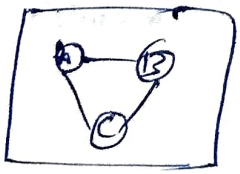
3) Communication :- Two (or) more elements operate on the same input data

Fig:-

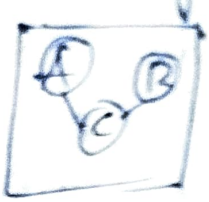


4) Procedural :- Elements of procedural Cohesion ensure the order of execution one after other

5) Temporal :- Elements are related by time involved (parallely (or) at a time).



6) Coincidental :- Elements are not related to any module hence they can be kept any where



7) Logical :- The elements are logically related but not functionally

IV) Coupling (min coupling)

Coupling is the measure of the degree of independence b/w the modules. A good S/W will have low Coupling.

Types

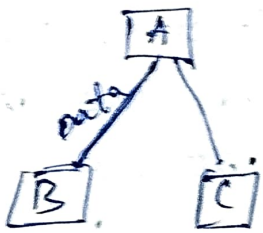
1) Data Coupling
2) Stamp "
3) Control "
4) External "
5) Common "
6) Content "

Best Low ~~High~~

~~Worst~~ High

1) Data Coupling :- When data of one module is passed to another module, this is called as Data Coupling.

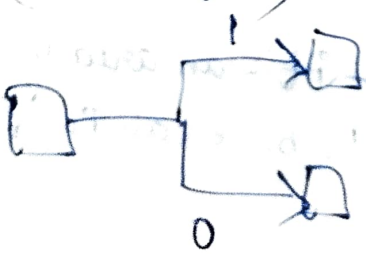
Figure



2) Stamp Coupling :- In this the Complete data structure is passed from one module to another module.

Figure

3) Control Coupling :- If the modules communicate by passing control information then they are said to be Control Coupling. (like flags)

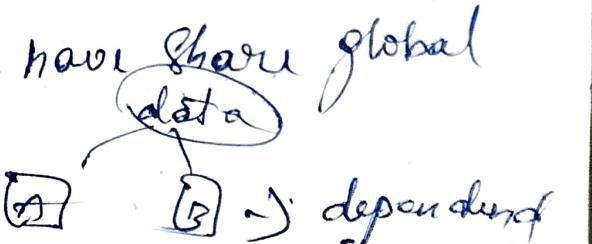


'1' is executed
'0' is not executed

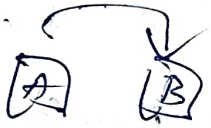
Single flag.

4) External Coupling:- In external coupling the module depends on other modules external to the s/w (like DB / server).

5) Common Coupling:- The module have share global data structure / global data.



6) Content Coupling:- Here one module modifys data of another module.



* Conducting Component level design ...

→ Component level design is elaborative in nature.

→ The designer must transform information from the analysis and architectural model into a design representation that provides sufficient details to guide the construction (coding & testing) activity.

The following steps represent a typical task set for Component-level design, when it is applied for an object-oriented sy.

Step ① Identify all design classes that correspond to the problem domain

→ Using the requirements and architectural model, each analysis class and architectural component is elaborated.

→ All design (or) classes that correspond to the problem domain are identified.

Problem domain - it is an area of expertise (or) application that needs to be examined to solve a problem.

Step② Identify all design classes that correspond to the infrastructure domain. (Identify infrastructure design class)

→ All design class that correspond to infrastructure domain are identified.

→ These classes are described in the requirements model and are often missing from the architectural model, but they must be described at this point.

→ Classes and components in this category include..

→ ① GUI components (often available as reusable components),

→ ① operating s/y components → ② Object & data management components.

Step③ Elaborate all design classes that are not acquired as reusable components. (Elaborate design classes)

→ Specify message detail, when class or components collaborate

→ Identify appropriate interface for each component.

→ Elaborate attributes & define data structures required to implement them.

→ Describe processing flow within each operation in details.

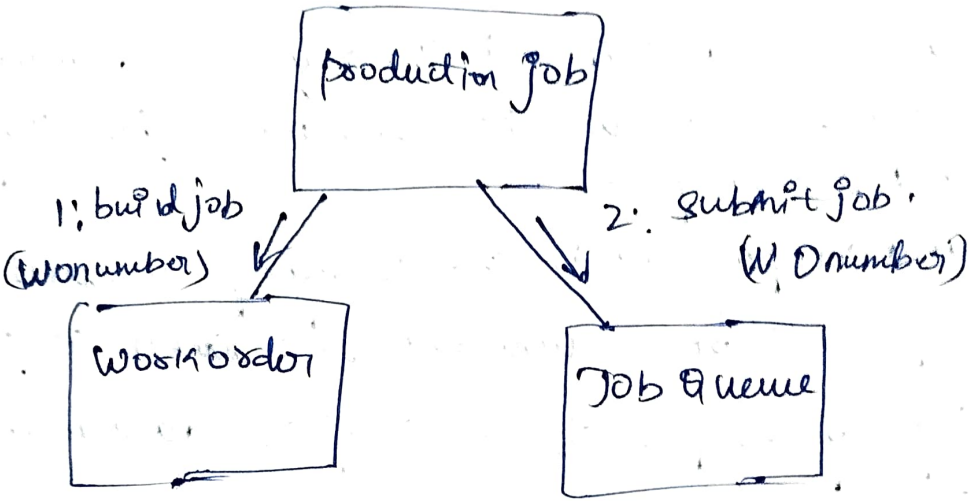
Step③a Specify message details when classes (or) components Collaborate. (collaboration details)

→ The requirements model makes use of a collaboration diagram to show how analysis classes collaborate with one another.

→ message can be elaborated by expanding this syntax in the following manner.

(guard condition) sequence expression (return value) := message name (argument list).

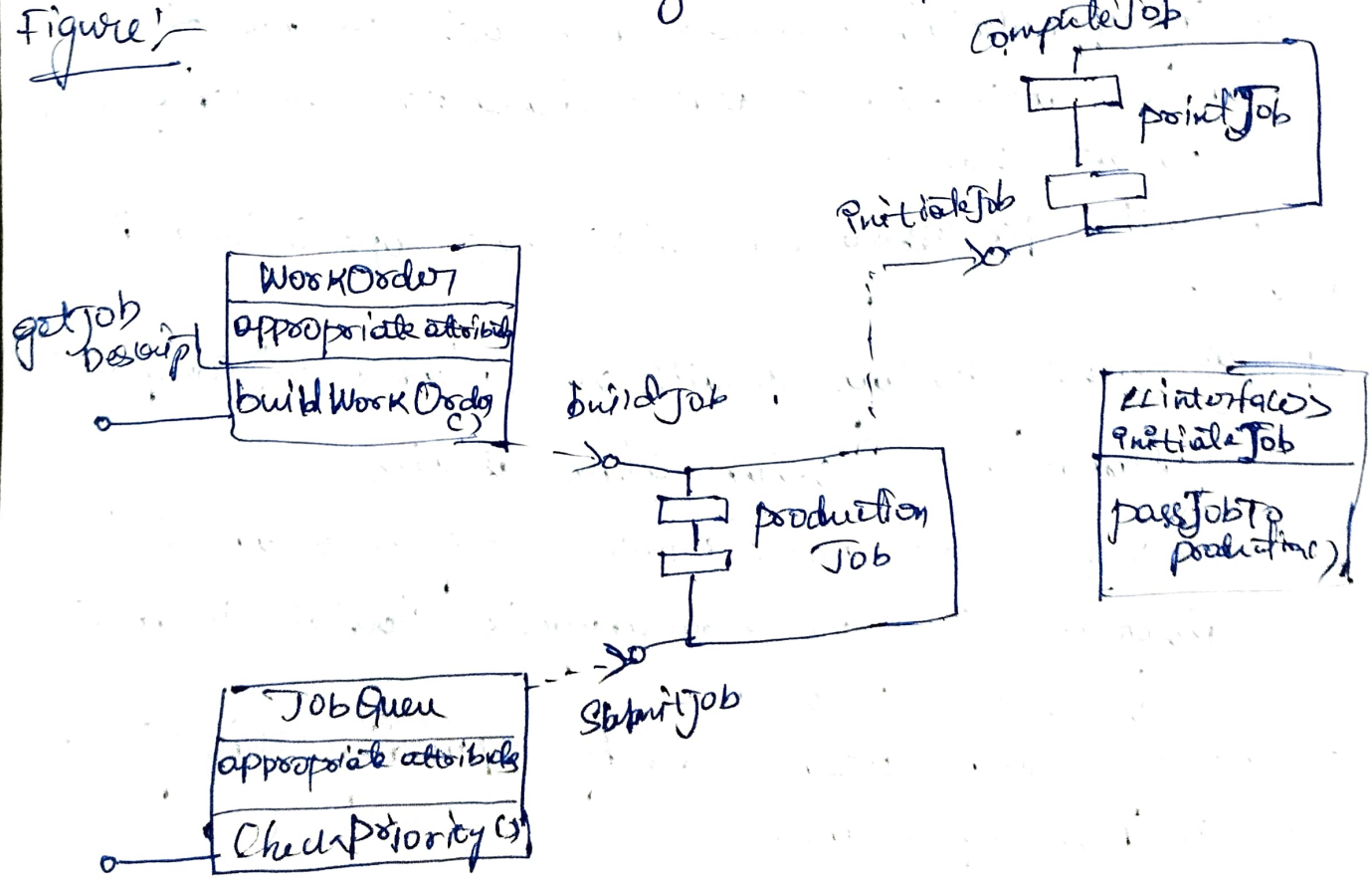
Figure:- Collaboration diagram with messaging.



Step 3b Identify appropriate interfaces for each component (appropriate interface)

Print job interface "Initiate job", which doesn't exhibit sufficient cohesion because it performs three different sub functions (refactoring can be performed).

Figure:-



Step (3c): Elaborate attributes and define data types and data structures required to implement them (Elaborate attributes)

→ In general, data structures and types used to define attributes are defined within the context of the programming language that is to be used for implementation.

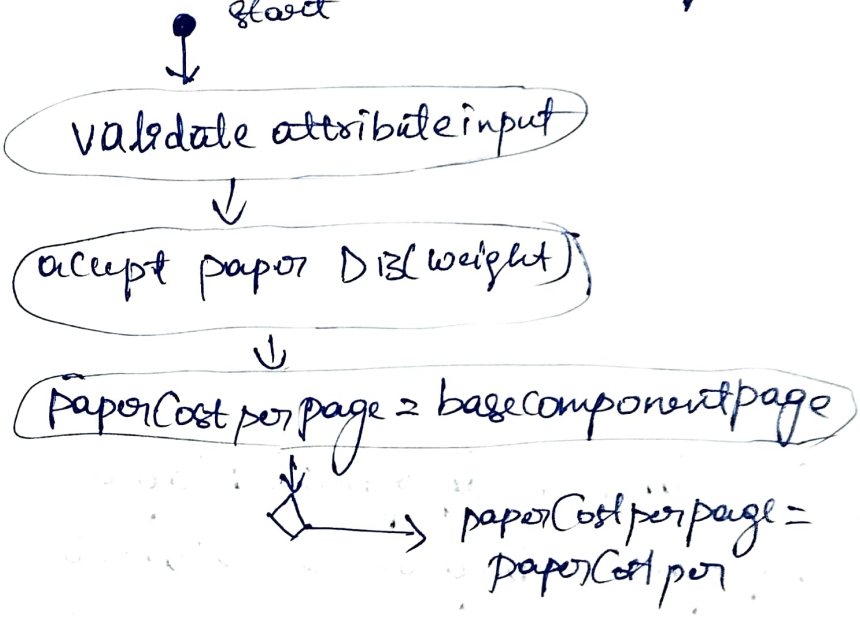
→ UML defines an attribute's data type using the following syntax:

name: type-expression initial-value {property string}

Step (3d): Describe processing flow within each operation in detail. (Describe processing flow)

→ The process flow is described in detail using activity diagram.

Figure:-

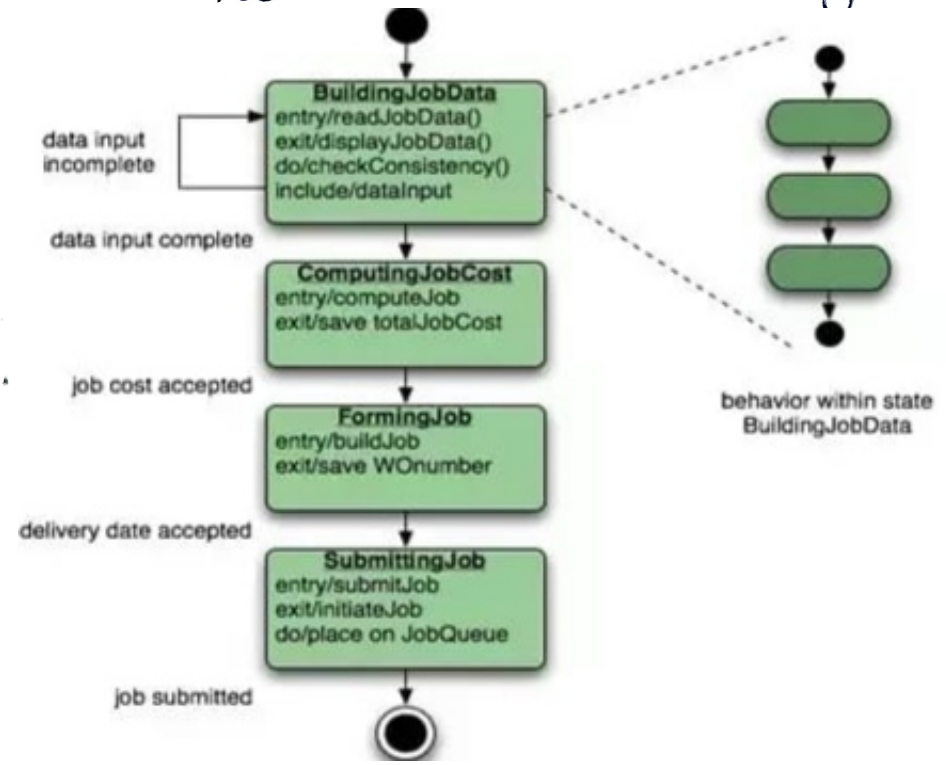


Step(4) persistance data sources

The persistance data sources are described (db + files) & the classes required to manage them are identified.

Step(5) Elaborate behaviour

→ It is sometimes necessary to model the behavior of design class. Transition from state to state having the form



Step(6): Elaborate deployment Diagrams

→ Deployment diagram are elaborated to represent the location of key package of component.

Step(7):- (Redesign/re consider)

The first Component-level model you creation will not be as complete, accurate (or) Consistent as the nth iteration you apply to the model.

* Golden rules – User Interface

Theo mandel trains 3 golden rules.

1) Place the user in control. I
2) Reduce the user's memory load II
3) Make the interface consistence III

→ These golden rules actually form the basis for a set of user interface design principles that guide this important aspect of s/w design.

User Interface design :- User interface design begins with the identification of user, task and environment requirements. Once user tasks have identified, user scenarios are created and analyzed to define a set of interface objects and actions.

I) Place the user in control

→ Define interaction in such a way that the user is not forced into performing unnecessary actions
→ provide for flexible interaction.
→ User interaction to be interruptible and reversible.
→ Hide the technical internal from the casual user

II) Reduce user memory load :-

→ Reduce demands on users short team memory
→ establish meaningful defaults.
→ Define shortcuts.
→ Disclose information in a progressive fashions.

III. Make interface consistent

→ Allow user to put the current task into a meaningful context.
→ Maintain consistency across all applications.
→ use the past interaction models.

* User Interface design

→ UI design creates an effective communication b/w a human and a computer.
→ It is the front end application view to which user interacts in order to use the S/W.
→ User can manipulate and control S/W and H/W by means of UI.
Eg:- Computers, mobile phones, cars, Airplanes, music players etc.
→ It provides a platform for human-computer interaction.
→ The S/W becomes most popular if its user interface is :-

* Attractive
* Simple to use
* responsive in short time
* Clear to understand
* Consistent on all interfacing screens.

→ UI is broadly divided into two categories

1) Command line interface (CLI)
2) Graphical User Interface (GUI)

1) Command Line Interface (CLI)

→ CLI has been a great tool to interact.
→ The user needs to remember the syntax of command and its use.
→ less amount of computer resources as compared to GUI
→ CLI provides a command prompt, where the user types the command and feeds to the system.

CLI elements

a) Command prompt
b) Cursor
c) Command.

2) Graphical User Interface (GUI)

→ GUI provides the simple interactive interface to interact with the system.

→ GUI can be a combination of both h/w and s/w.

GUI elements

1) Window — Application window
2) Tabs — Dialogue Box
3) Menus — Text-Box
4) Icons — Buttons.
5) Cursor

* User Interface Design models

→ UI analysis and design process is an iterative process & it can be represented as a spiral model.

It consists of 4 frame work activities

1) User, task, & environment analysis, & modeling :-

→ From each category requirements are gathered.

→ Based on the requirements developer understand how to develop the interface.

→ Once all the requirements are gathered a detailed analysis is conducted.

→ The analysis of user environment focuses on the physical work environment.

2) Interface design :-

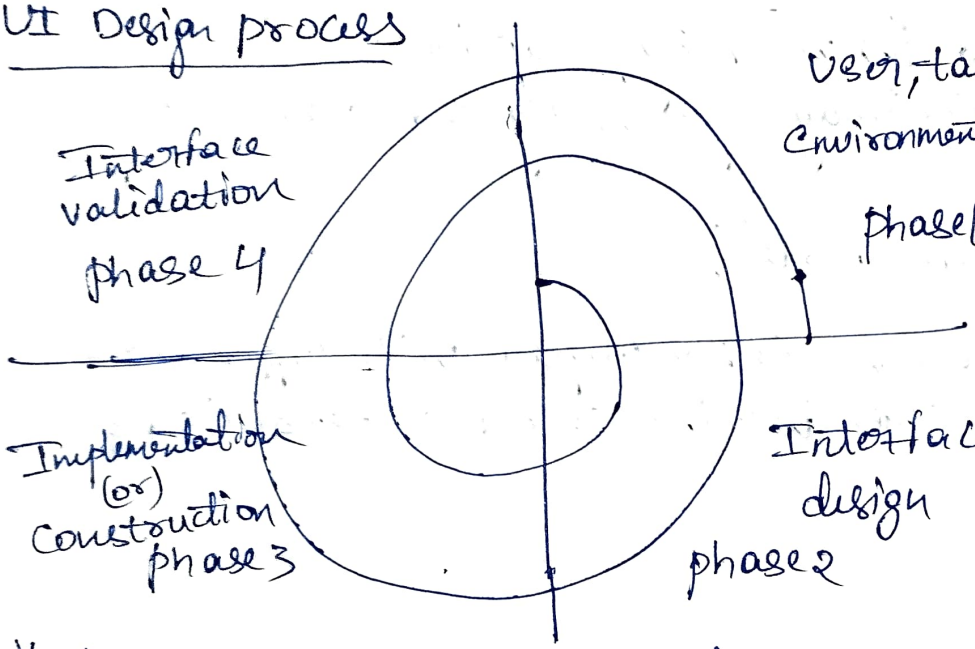
→ The goal of this phase is to define the set of interface objects and actions i.e. control mechanisms that enable the user to perform desired tasks.

→ This phase serves as the foundation for the implementation phase.

3) Interface construction:- UI development tools may be used!
=> As iterative design process continues a UI toolkit that allows the creation of windows, menus, device interaction, error messages, commands etc.

4) Interface validation:- Correctness of s/y is validated.
-> It should be able to perform tasks correctly.
-> It should be able to handle a variety of tasks.
-> It should achieve all the user's requirements.
-> It should be easy to use and easy to learn.

UI Design process



* User Interface Analysis and Design

-> The overall process for analyzing and designing a UI begins with the creation of different model of system function.

4 models

1) User model (established by human engineer (or) s/w engineer)
-> Establish the profile of the end-user of the s/y (Based on age, gender, education background & various other considerations)
-> This is an important criteria since, it is the end user who is going to deal with the product forever after developed by a given s/w development team.

3 categories

a) No users (new & inexperienced)
b) Knowledgeable, intermediate users.
c) Knowledgeable, frequent users.

2) Design model :- (Created by a S/w engineer)

-> It is derived from analysis model of requirements & also based on users of the system.

3) Users Mental model :- (Developed by users when interacting with the application).

-> often called the users sys perception.
-> Consists of the image of the sys that users carry in their hands.

4) Implementation model :-

-> The interface look & feel coupled with supporting information that describes interface semantic & syntax.

Cohesion

→ Cohesion is a concept of intra module
→ It represents relationship within a module.
→ Max. Cohesion is important.
→ It is a measure of the degree to which the elements (or) components of the module are functionally related.
→ A good S/w design will have high Cohesion.
→ It functional strengths of modules.

Types

1) Functional cohesion
2) Sequential "
3) Communication "
4) Procedural "
5) Temporal "
6) Coincidental "
7) Logical

High ↑ best
Low ↓ worst.

Coupling

→ It is a concept of inter-module
→ represents relationships b/w the module.
→ Min. Cohesion is negligible.
→ It is a measure of the degree of independence b/w the modules
→ A good S/w design will have low coupling.
→ It represents independence among modules.

Types

1) Data Coupling
2) Stamp "
3) Control "
4) External "
5) Common "
6) Content ".

Low ↑ ~~worst~~
~~best~~ High