Software Engineering?

Unit 3

Software Design

Design concepts : design process - design concepts - modularity, coupling and cohesion - Design model - modeling principles; structured design; architectectural design : Architectural styles ; Architecture for Network based Applications - Decentralized architectures.

* Software Design Process

→ Design creates a representation or model of the software. The design model provides details about:

(i) software architecture

(ii) data structures

(iii) interfaces

(iv) components to implement the system

→ All the stakeholder requirements, business needs and technical considerations all come together in the formulation of the product.

Who does software design? - software engineers

Why is software design important? - allows to model the system or product that is to be built

→ The model can be assessed for quality & improved before

- code is generated
- tests are conducted
- end users are involved in large numbers

What are the steps in software design?

1. Represent the architecture of the system / product
2. Model interfaces that connect the software to end users
3. Software components are designed

What is the work product of software design?

The primary design product is a design model that encompasses architectural, interface, component-level and deployment representations.

How does one know if the software design is right?

→ The software team assesses the design model by checking:

(i) whether it contains, errors, inconsistencies or omissions

(ii) whether there are better alternatives

(iii) whether the model can be implemented within the constraints schedule and cost that have been established.

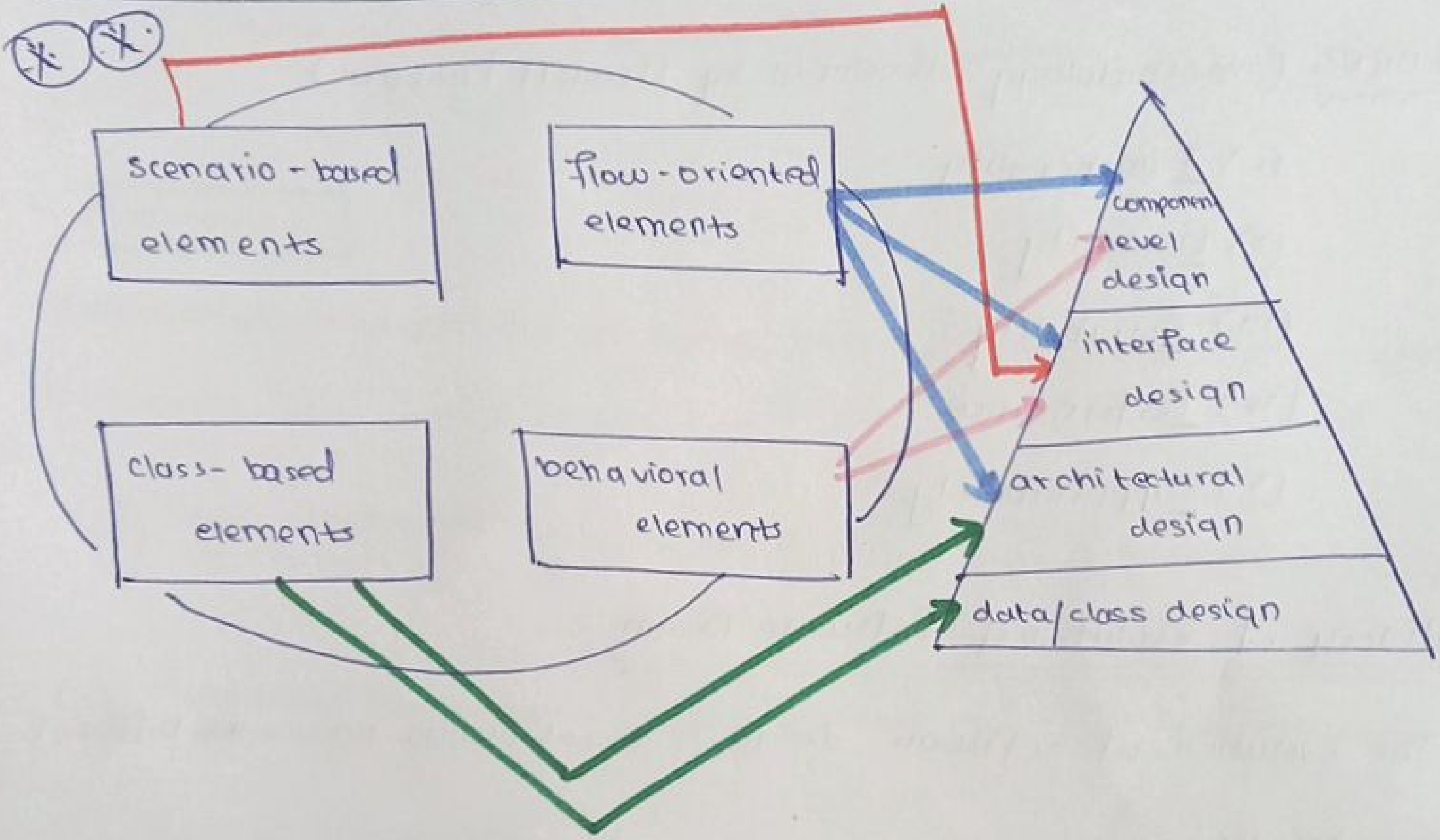
* Phases in the Design Process

A Diversification : acquire a repertoire of alternatives

get raw materials - components, component solutions & knowledge

B. Convergence : choose and combine elements from the repertoire to meet the design objectives

* Relation of Analysis to Design



* Design Quality Guidelines

A. General Guidelines : (i) should have an architectural structule

(ii) modularity

(iii) have distinct representations of data, architecture, interfaces and components

(iv) have independent functional units

(v) interfaces should reduce complexity

(vi) design should be from a repeatable method

(vii) notation should effectively communicate the meanings

B. McGlaughlin's Guidelines :(i) must enable all requirements

(ii) must be readable & understandable

(iii) should address data, functional and behavioral domains

C. FURPS (methodology developed by Hewlett Packard)

(i) Functionality

(ii) Usability

(iii) Reliability

(iv) Performance

(v) Supportability

* History of Evolution of Software Design

→ The evolution of software design is a continuous process. ~~the~~ Different approaches are:

(i) Procedural Approach

(ii) Object Oriented Approach

(iii) Aspect - Oriented Approach

(iv) Model- Driven Development

(v) Test -driven development

* Design Concepts

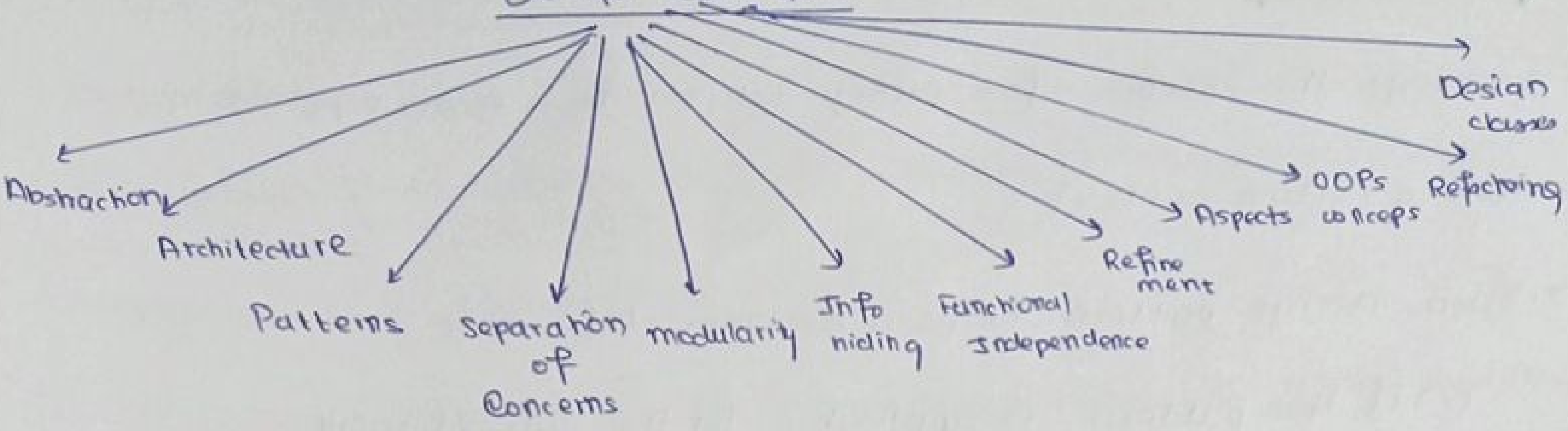
→ uniform design

→ should accommodate change

→ should minimize coupling between modules

→ should have graceful degradation

Design Concepts



A. Abstraction

→ concentrate on a problem at some level of generalization w/o regarding irrelevant low level details

(i) Data Abstraction – collection of data that describes a data object

(ii) Procedural Abstraction - each instruction has a limited function

(iii) Control Abstraction - program control mechanism without specifying internal details - eg. semaphore, rendezvous

B. Software Architecture

Architectural design should deal with the following:

(i) Structural Properties – components of a system & their interaction with other components

(ii) Extra Functional Properties – addresses how architecture achieves requirements for performance, reliability & security

(iii) Family of Related Systems – ability to reuse architectural building blocks

C. Patterns

→ conveys the essence of a proven solution to a recurring problem within a certain context

→ Each design pattern helps a designer determine

(i) if the pattern is applicable for the current work

(ii) if the pattern can be reused

(iii) whether the pattern can serve as a guide for developing functionally | structurally different pattern

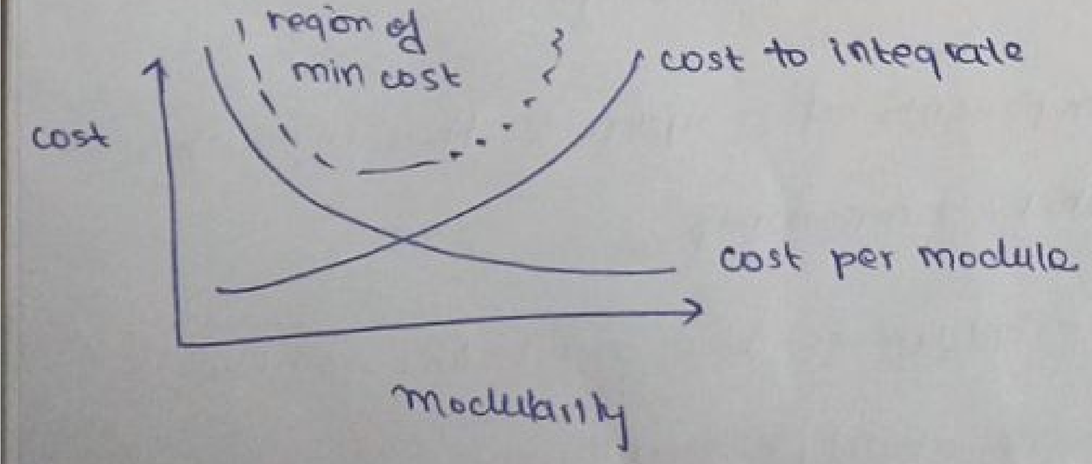
D. Separation of Concerns

→ A complex problem can be easily handled if it is subdivided into pieces that can be solved and/or optimized easily.

E. Modularity

→ software is divided into separately named and addressable components.

→ Follows a divide and conquer approach.



→ Modularity has the following sub-principles:

(i) Modular Decomposability - provides a systematic method to decompose a problem to subproblems

(ii) Modular Composability - enable reuse of existing components

(iii) Modular Understandability - whether the module can be understood as a stand-alone unit.

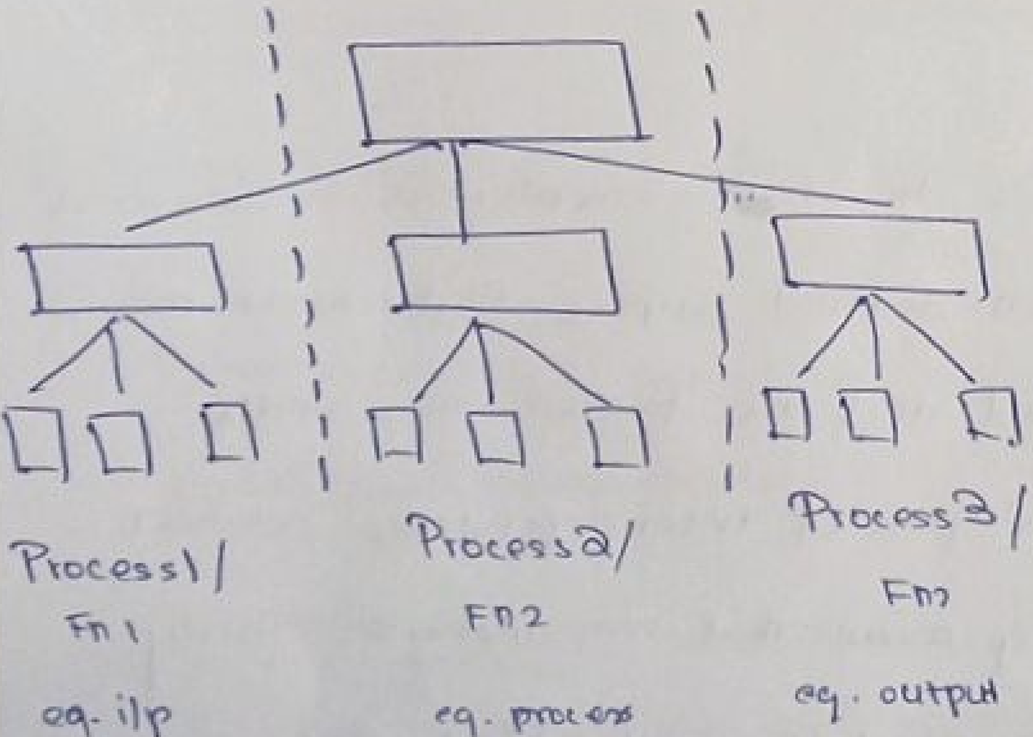
(iv) Modular Continuity - if changes are made in individual modules, the impact ~~of~~ the overal side effects are reduced

(v) Modular Protection - errors ~~or~~ in modules are localized to that module alone.

→ Modularity can be represented by means of structural diagram;

Types of Structural Partitioning

A. Horizontal Partitioning



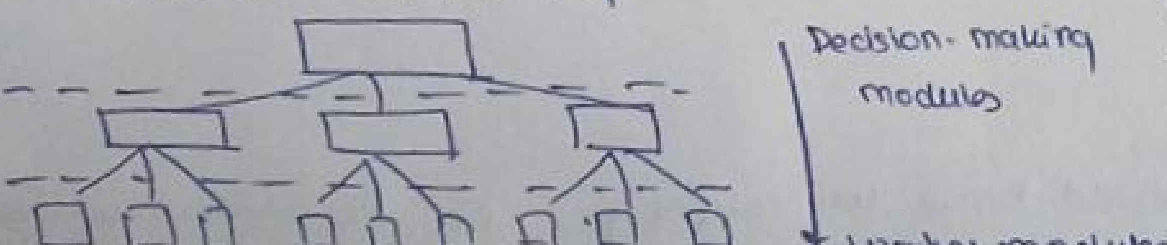
→ easier to test

→ sometimes easier to maintain

→ sometimes fewer side effects propagate

→ easier to add new features

B. Vertical Partitioning



→ control & work modules are distributed top down

→ Top level modules perform control functions

→ Lower level modules perform computations - less side effects very maintainable

F. Information Hiding

→ Modules are characterized by design decisions that are hidden from others

→ Modules communicate only through well-defined interfaces

→ Helps enforce access constraints

→ Helps accommodating change & reducing coupling

G. Functional Independence

→ Critical in dividing system into independently implementable parts

→ Measured by 2 qualitative criteria:

① Cohesion - relative functional strength of a module

② Cohesion - relative interdependence between modules

* Cohesion and Coupling

Cohesion refers to the degree to which elements in a module work together, to fulfil a single well-defined purpose. High cohesion means that elements are closely related and are focused on a single purpose. Coupling refers to degree of interdependence between software modules. High coupling means that modules are closely connected and changes in one module may affect other modules.

A. Coupling

→ Good software should have low coupling, i.e low interdependence between modules.

→ There are different kinds of coupling, which is ranked as follows:

BEST

Data coupling
↓
Stamp coupling
↓
Control coupling
↓
External coupling
↓
Common coupling
↓
content coupling WORST

(i) Data coupling - if communication between modules is only in the form of passing data

(ii) Stamp coupling - data structures are passed between modules

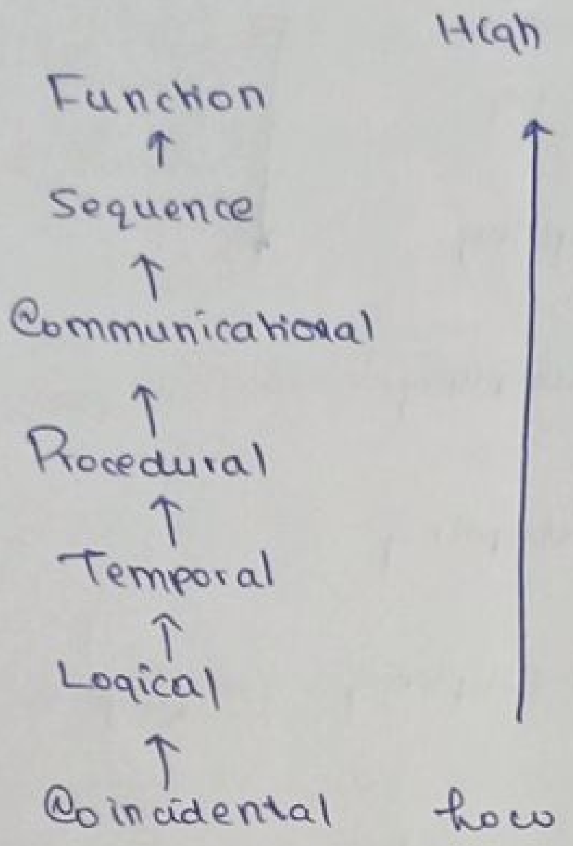
(iii) Control coupling - modules communicate by passing control info - like say a sort function

(iv) External coupling - modules depend on other module, external to the software being developed

(v) Common coupling - ~~one module can modify data of another module~~ modules have shared data as global data structures

(vi) Content coupling - one module can modify the data of another module.

B. Cohesion - Cohesion is a measure of the degree to which the element of the module are functionally related. Levels of cohesion based on their types are as follows:



(i) Functional cohesion - every essential element for a computation is present in a single component

(ii) Sequential cohesion - data flows sequentially from one module to another

(iii) Communicational cohesion - 2 elements operate on the same input data or contribute to the same output data.

(iv) Procedural cohesion - ensures elements are executed in order - actions not reusable

(v) Temporal cohesion - elements are related by their timing involved

(vi) Logical cohesion - elements are logically related - not functionally

(vii) Coincidental cohesion - elements are not related, accidental

H. Refinement

→ a process by which one or several instructions of the program are decomposed into more detailed instructions

→ step wise refinement is a top-down strategy

→ The designer is forced to develop low level details as the design progresses.

→ Note that abstraction & refinement are complementary concepts

I. ASPECTS

→ An aspect is a crosscutting concern. For eg. if there are two requirements A and B, requirement A crosscuts requirement B, if a software (refinement) decomposition has been chosen where B cannot be satisfied without taking A into account. (eg. validation must happen before given access to a registered user)

J. Refactoring (CAT 2-Q)

→ a reorganization technique that simplifies the design of a component without changing its function or behavior

K. OOPs Design Concepts

→ includes usage of concepts like

- classes and objects
- inheritance
- messages
- polymorphism

* Design Classes

→ There are 5 different types of design classes

A. User Interface Classes - deals with HCI - human computer interaction

B. Business Domain Classes - identify attributes and services required to implement some element of the business domain.

C. Process Classes - implement lower-level business abstractions

D. Persistent Classes - represents data stores (database) that exist beyond the execution of the software.

E. System classes - implement software management to enable the system to operate & communicate within its computing environment & with the outside world.

* Four characteristics for a well-formed design class

(i) complete and sufficient

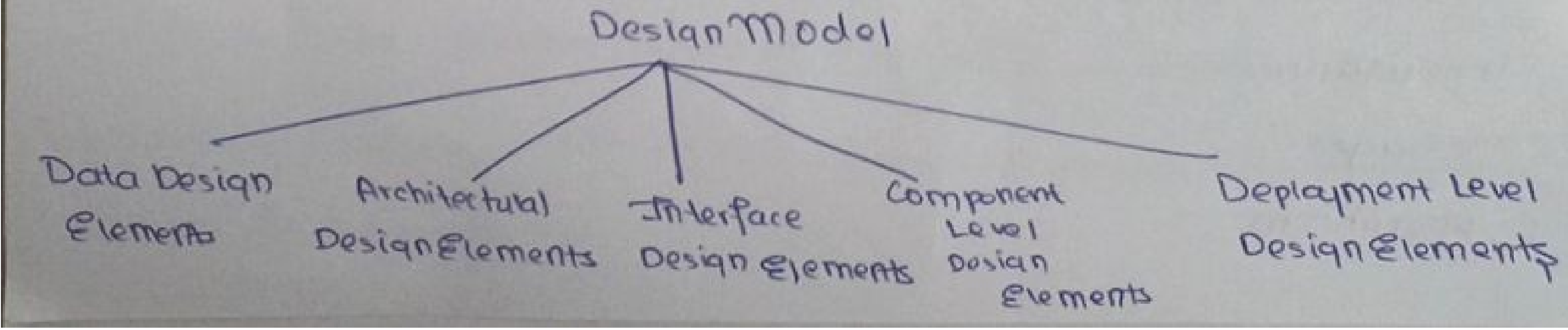
(ii) Pimitiveness - (each method accomplishes one task)

(iii) high cohesion

(iv) low coupling

* Design Model

→ The design model is classified into the following elements


Design Model
Data Design Elements
Architectural Design Elements
Interface Design Elements
Component Level Design Elements
Deployment Level Design Elements

A. Data Design Elements

at the component level → design data structures and associated algorithms

at the application level → translation of a data model into a database

at the business level → collection of info. stored in disparate databases and reorganizing into a data warehouse.

B. Architectural Design Elements

→ gives an overall view of the software

→ The architectural model is derived from 3 sources:

(i) info. about the application domain

(ii) data flow diagrams

(iii) availability of architectural styles and patterns

C. Interface Design Elements

→ describes how the software communicates with itself, to other systems and with humans

→ has 3 components:

(i) user interface

(ii) external interfaces to other systems

(iii) internal interfaces between various design components

D. Component Level Design Elements

→ describes the internal detail of each software component

→ defines data structures for all local data objects and algorithmic detail for all processing

E) Deployment ~~of~~ Level Design Elements

→ indicate how software functionality and subsystems will be allocated within the physical computing environment.

* Architecture Design

→ Architectural design represents the structure of data & program components that are required to build a computer-based system.

→ It considers:

(i) architectural style

(ii) structure & properties of the components

(iii) inter-relationships among all the components

Who does architecture design? - software engineers

or

data warehouse designer - data architecture

system architect - selects an appropriate architecture from software requirement analysis

Why is architecture design important?. - provides the big picture & ensure that ~~the~~ one has built the system right.

→ analyze effectiveness of design

→ reduce risks associated w/ the construction of the software.

→ enables communication btwn all stakeholders

What are the steps in architecture design?

(i) data design

(ii) make architectural structure

(iii) analyze alternate architectural styles

(iv) elaborate on chosen architecture

What is the work product of architecture design?

→ An architecture model encompassing data architecture and program structure

→ component properties and relationships

How does one know If the architecture design is right?

At each stage, review for : clarity
correctness
completeness
consistency

* Architectural Descriptions – a collection of products to document an architecture (JEEE standard)

* Architectural Genres

→ defines a specific category within the overall software domain

A. Artificial Intelligence - systems that simulate human cognition, locomotion

B. Commercial or Non-Profit - systems that are fundamental to the operation of a business operation

C. Communications - systems that provide the infrastructure for transferring and managing data

D. Content authoring - systems to create text/ multimedia artifacts

E. Goverment - systems that conduct the operations of any kind of govt. / political entity

F. Industrial - systems that simulate or control physical processes

* Architectural Styles

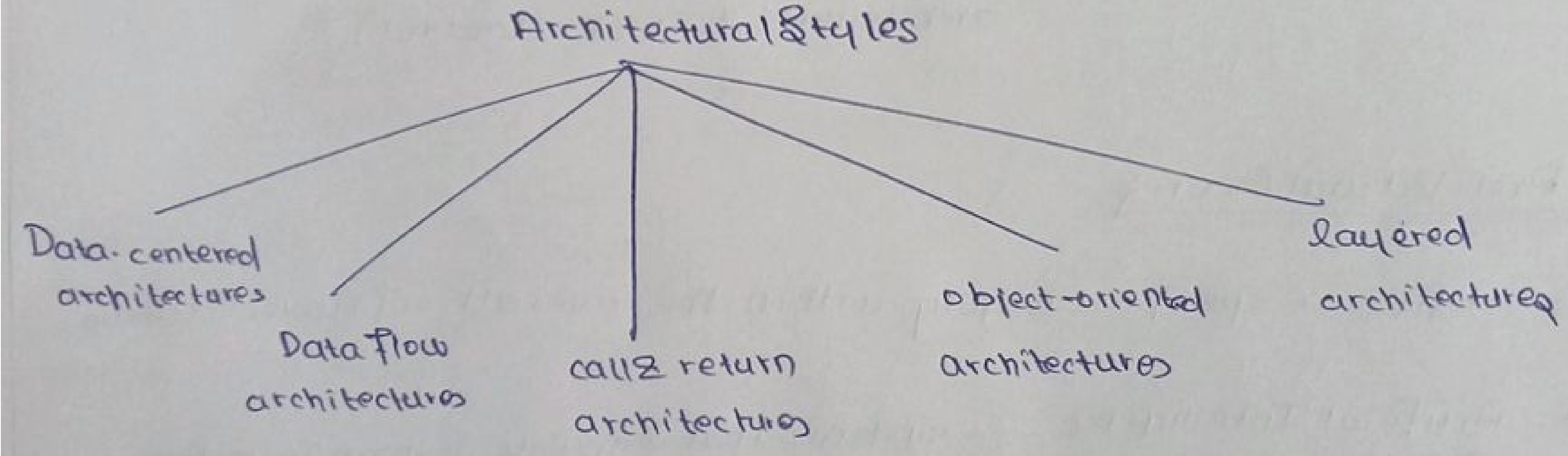
→ Each architectural style describes :

(i) a set of components

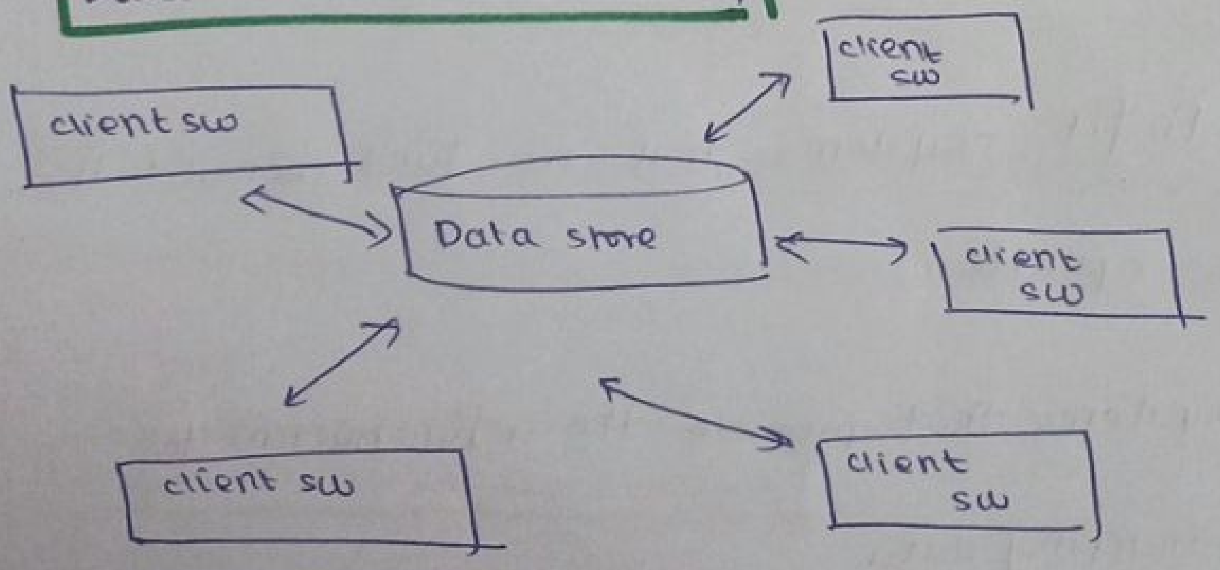
(ii) a set of connectors

(iii) constraints

(iv) semantic models

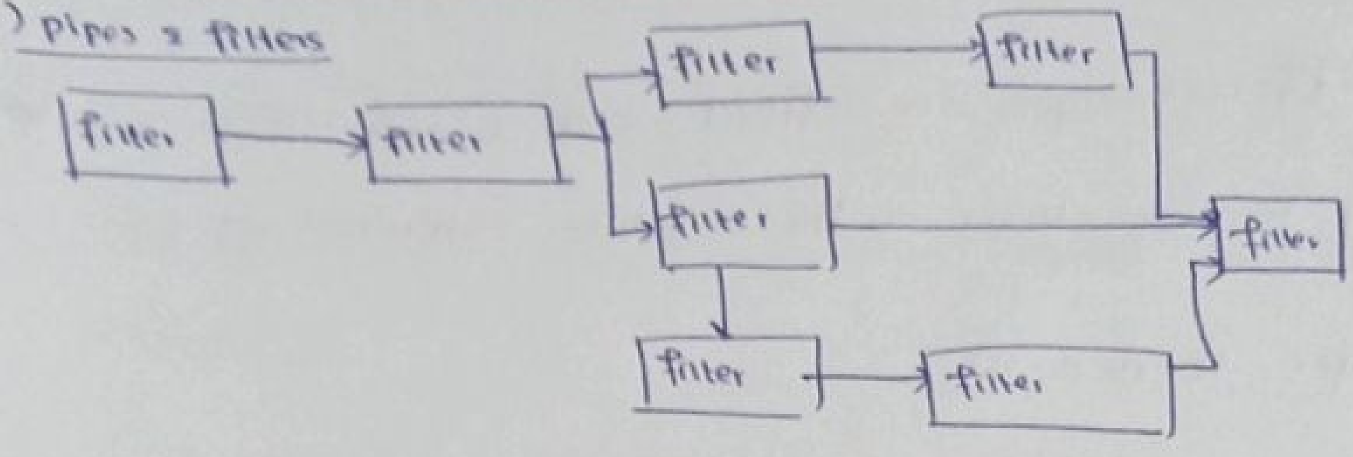

Architectural Styles
Data-centered architectures
Data flow architectures
call & return architectures
object-oriented architectures
layered architectures

A. Data - Centred Architectures

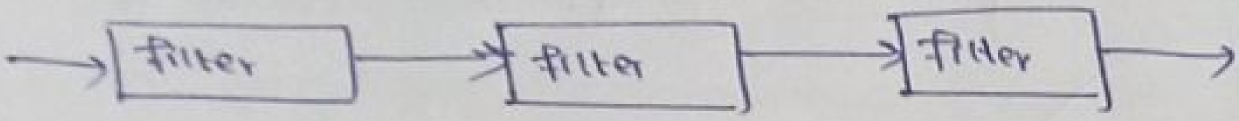


B. Data Flow Architecture.

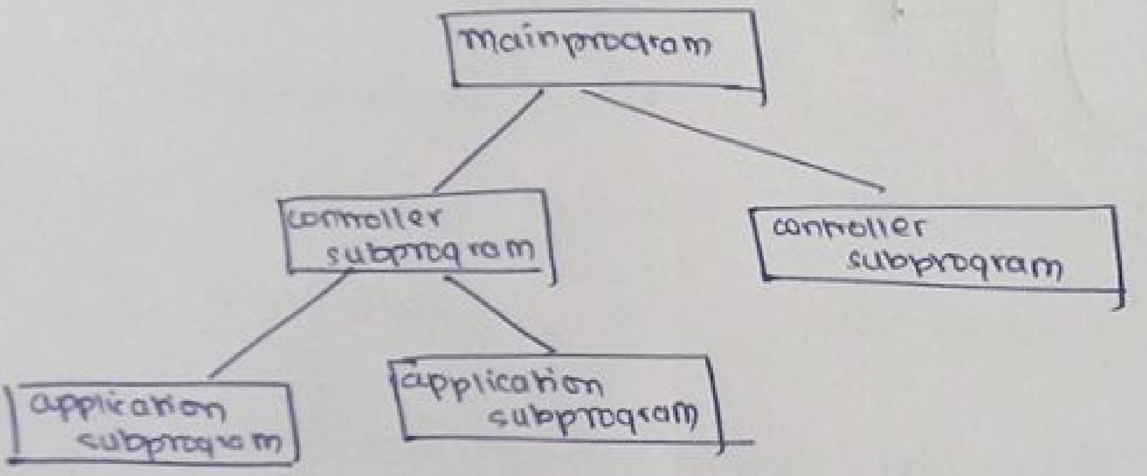
(i) pipes & filters



(ii) batch sequential



C. Call and Return Architecture



can be of 2 types:

(i) Main program / subprogram architectures - main program invokes a number of program components which may invoke other component

(ii) Remote procedure call architecture - components of a main program/ sub program architecture are distributed across multiple computers on a network

D. Object - Oriented Architecture

→ encapsulate data and operations that must be applied to manipulate the data

→ communication & coordination is via message passing

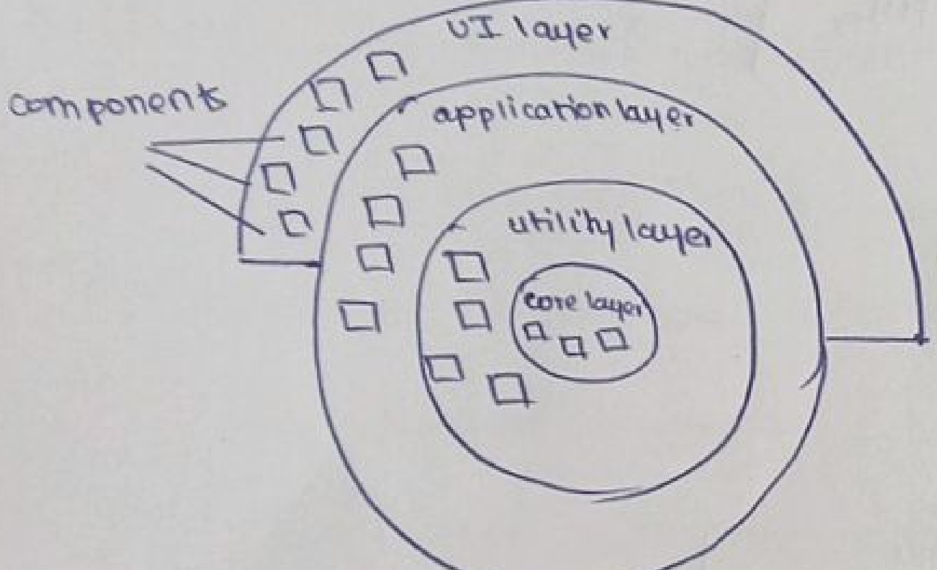
E. Layered Architecture

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

→ At the outer layer, there is UI

→ At the inner layer, components perform operating system interfacing

→ At immediate layers: utility services and application software fns



* Architectural Patterns

A. Concurrency - must handle multiple tasks in a manner that simulates parallelism

B. Persistence - data persists if it survives past the execution of the process that created it. There are 2 patterns:

(i) a DBMS

(ii) an application level persistence pattern

C. Distribution - systems or components communicate with one another in a distributed environment

broker - acts as a middle-man between the client & server component.

* Architectural Design

Components / Key Features

1. The design should define the external entities and the nature of the interaction

2. Architectural archetypes should be defined. Archetypes is an abstraction that represents one element of system behavior.

3. Refine software components that implement each archetype.

Systems interacting with Target System

Systems that interoperate with the target system are represented as:

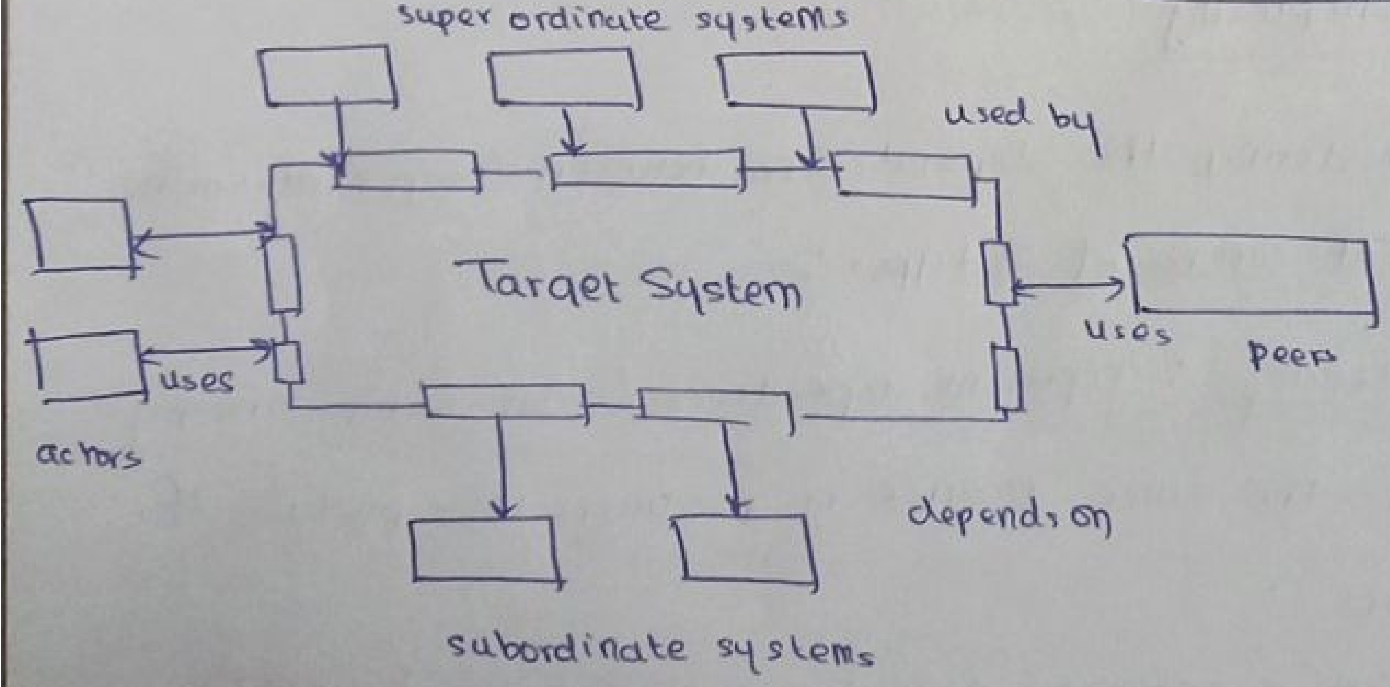
(i) superordinate systems - part of a higher level processing scheme

(ii) subordinate systems - provide data or processing

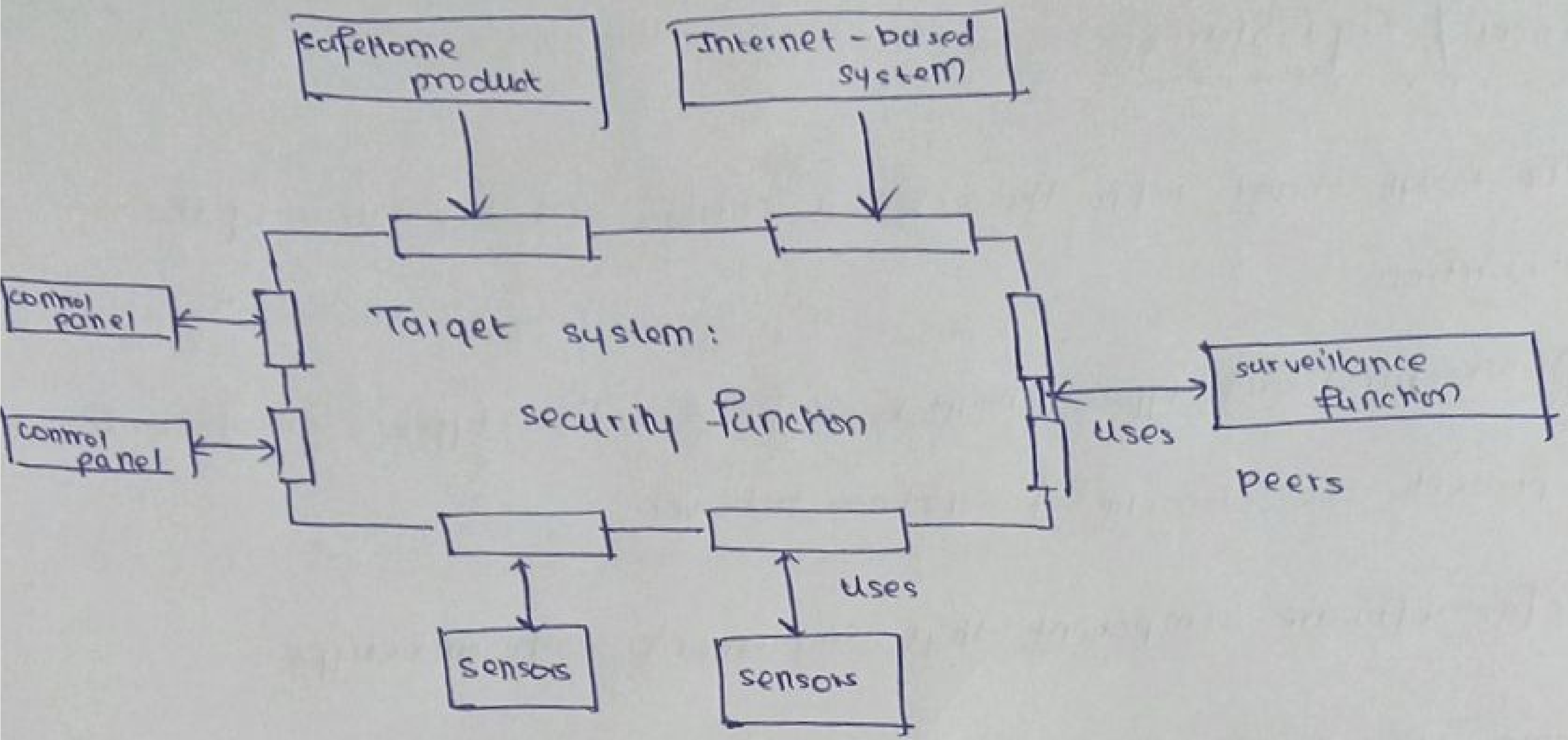
(iii) peer-level systems - info. is produced / consumed by peers

(iv) actors - entities - people / devices that interact with the target system.

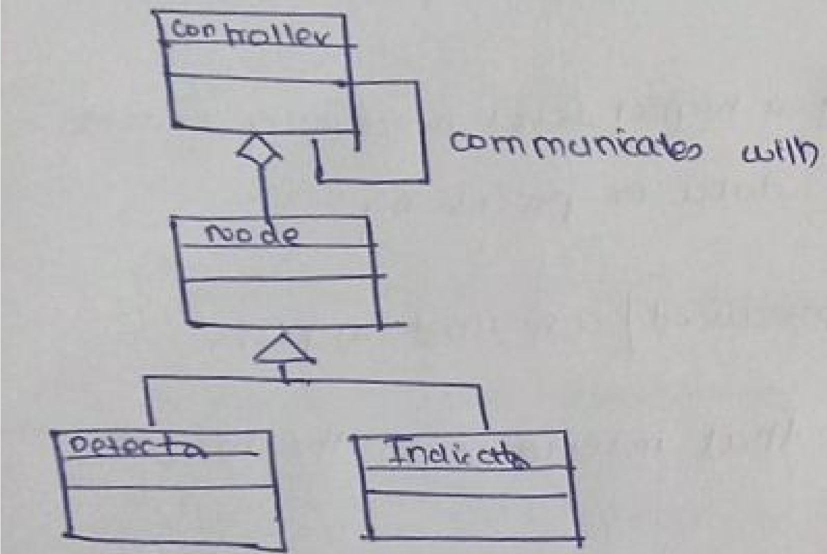
These systems are represented in an Architectural Context Diagram (ACD) as:



Example: The ACD diagram for a safe home security system would be:



Some of the archetypes of this system would be:



* Architectural Complexity

→ assessed by considering the dependencies between components within the architecture. It can be of 3 types:

(i) Sharing dependencies : represent dependence / relationships among customers who use the same resource or producers who produce for the same consumers

(ii) Flow dependencies : represent dependence relationships between producers and consumers of resources

(iii) Conshained dependencies : represent constraints on the relative flow of control among a set of activities

* ADL and Architectural Design Process

→ ADL or Architectural Description Language provides semantics & syntax for describing a software architecture

→ Provides the designer with the ability to:-

(i) decompose architectural components

(ii) compose individual components into larger architectural blocks

(iii) represent interfaces

Steps in ADL

1. Establish type of information flow
2. Indicate flow boundaries
3. Map data flow diagram into program
4. Define control hierarchy by factoring
5. Refine with ~~[illegible]~~ design heuristics
6. Refine and elaborate on architectural design.

Steps in Transform Mapping

- meant to map DFD to a specific architectural style

Steps
1. Review fundamental model
2. Review and redefine DFD
3. Identify if DFD has transform or flow characteristics

4. Isolate the transform center by ~~refactoring~~ specifying incoming & outgoing flow boundaries

5. Perform first-level factoring

6. Perform second-level factoring

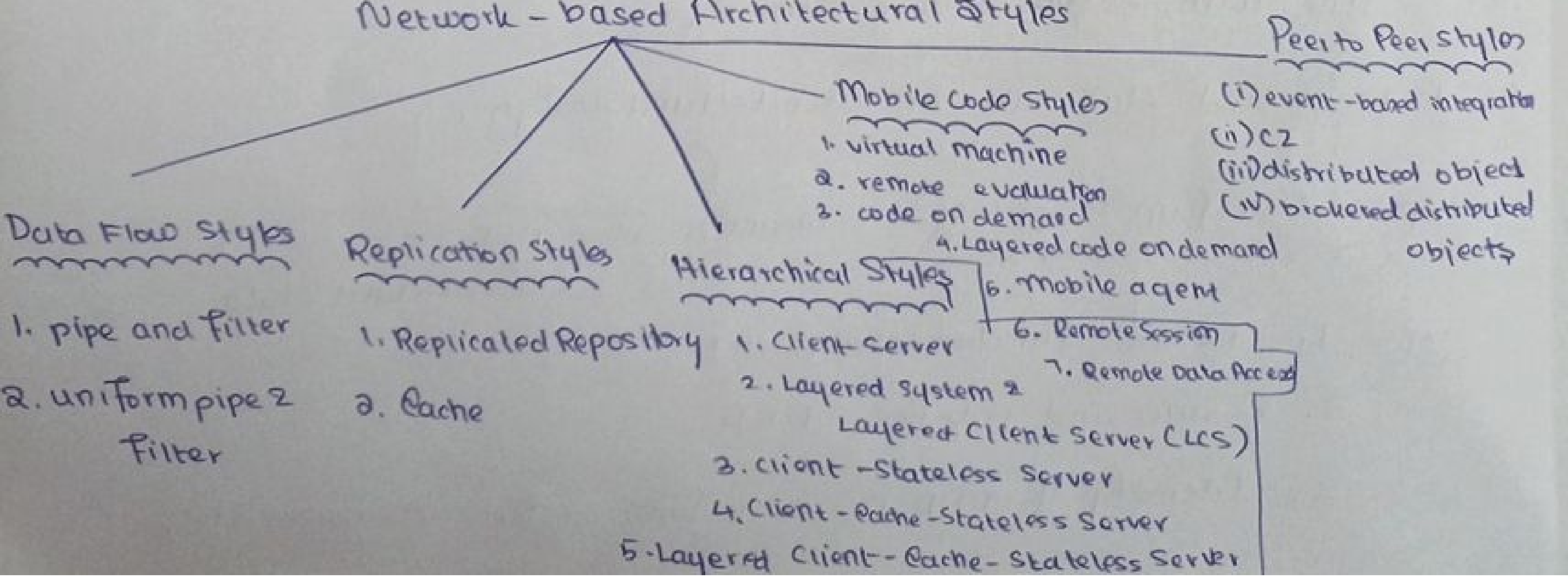
7. Refine first iteration architecture w/ design heuristics

(X)

* Differences between Transaction and Transform Mapping

Transaction Mapping	Transform Mapping
1. Ensures data consistency	1. May change data format and content
2. Doesn't manipulate data	2. Completely transforms data
3. Linear & Simple data flow	3. Complex data flow

* Network-based Architectural Styles


Network-based Architectural Styles

Data Flow Styles
1. pipe and filter
2. uniform pipe & filter

Replication Styles
1. Replicated Repository
2. Cache

Hierarchical Styles
1. Client-server
2. Layered System & Layered Client Server (LCS)
3. Client-Stateless Server
4. Client-Cache-Stateless Server
5. Layered Client-Cache-Stateless Server
6. Remote Session
7. Remote Data Access

Mobile Code Styles
1. virtual machine
2. remote evaluation
3. code on demand
4. Layered code on demand
5. mobile agent

Peer to Peer styles
(i) event-based integration
(ii) C2
(iii) distributed object
(iv) brokered distributed objects

A. Data - Flow styles

(i) pipe and Filter - each component (Filter) reads streams of data on its inputs and produces streams of data on its o/ps. by applying a transformation to the input streams & processing them incrementally.

disadvantage = propagation delay

(ii) uniform pipe and Filter - same as (i), but all pipes and filters must have the same interface.

disadvantage - may reduce performance if data needs to be converted.

B. Replication Styles

(i) Replicated Repository - improve accessibility by having more than one process provide the same service - eg. distributed file systems

(ii) Caching - replication of the result of an individual request such that it may be reused by later requests.

→ may be less useful than (i) because only recent requests will be stored, and the rest would be misses

C. Hierarchical Styles

(i) Client - Server - server offers services, clients send requests & the server either rejects the request or performs the request & sends the respons

(ii) Layered System and Layered Client-Server

↓ organized hierarchically, each providing services to the layer above it and uses the services of the layer below it., however adds over head

↓ same as client-server style, but adds proxy and gateway components

(iii) Client - Stateless - Server (CSS) - derived from client - server with the additional constraint that no session state can be maintained on the server. The client cannot take advantage of any stored context on the server.

(iv) Client -Cache - Stateless Server - same as (iii) but, add cache components

(v) Layered Client Cache Statless Server - derives from Layered client-server and client-cache stateless server, through the addition of proxy and/or gateway components

(vi) Remote Session - a variant of client-server that attempts to minimize complexity & maximize reuse of client components

(vii) Remote Data Access - a variant of client -server that spreads the application state across both client & server.

b. **Mobile Code Styles** - uses mobility to dynamically change the distance between the processing and source of data or destination of results.

(i) Virtual Machine

→ code is executed within a controlled environment to satisfy security and reliability concerns.

(ii) Remote Evaluation - derived from the client server and virtual machine styles.

→ The client component has the ~~know~~-how of how to perform a service but lacks resources - which are located at a remote site.

→ Server component at remotesite executes the code.

(iii) Code on Demand - client has resources - but lacks know-how on how to process them - sends a request to a remote server who has the ~~know~~-how

(iv) Layered Code on Demand Client Cache Stateless Server

add COD to LCCSS style

(v) Mobile Agent (MA) - entire computational component is moved to a remote site, along with its state, code, data

E **Peer to Peer Styles**

(i) Event-based integration - a component announces or broadcasts an event - all components invoke their components

(ii) C2 - directed at supporting large- grain reuse

combine event - based integration w/ layered client server

(iii) Distributed Objects - organizes a set of components as peers

- an object is an entity that encapsulate data, operations & procedures

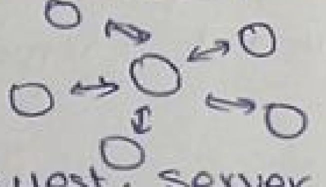
(iv) Brokered Distributed Objects -(iii) but with name resolution included.

* Centralized, Decentralized and Federated Architecture

Centralized Systems

→ uses client server architecture

→ one or more clients connected to a central server



→ client sends a request, server responds, highly dependent on network connectivity

→ run on a single computer system

→ eg. wikipedia

→ has a central clock, also has dependent failure of components (central node failure)

→ only vertical scaling is possible (can't scale after a limit)

→ bottlenecks can appear w/ high traffic (can have DOS)

→ no graceful degradation

Decentralized Systems

→ every node makes its own decisions

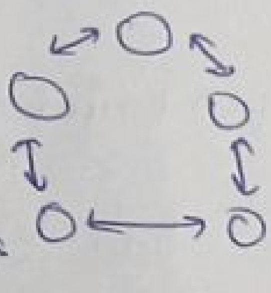
→ final behavior is an aggregate of the decisions of individual nodes

→ eg. bitcoin

→ lacks a ~~central~~ global clock

→ can have multiple central units

→ dependent failure of components



→ can scale vertically

→ may cause problems of coordination

→ no way to regulate node behavior

→ minimum problem of bottlenecks

→ increased transparency

→ high availability

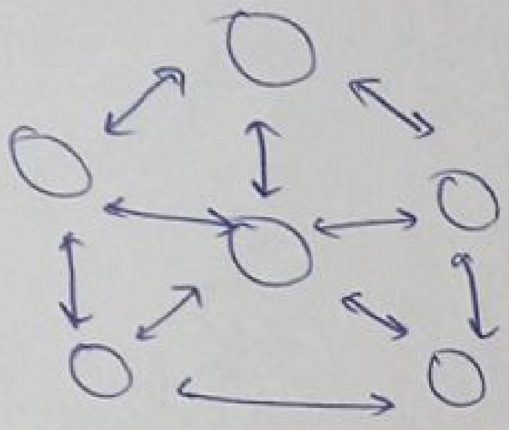
→ improved scalability

Federated Architecture

→ semi-autonomous de-centrally organized entities

→ There are local leaders with centralized support

→ cooperation between domains



Distributed Architecture

→ a collection of computer programs that utilize computational resources across multiple, separate computer nodes

→ separate nodes communicate & synchronize over a common network

→ These dist. systems aim to remove bottlenecks & central points of failure

eg. Google search engine

Characteristics

- → resource sharing
- → simultaneous processing
- → scalability
- → error detection
- → transparency

→ can be homogenous or heterogenous systems