Unit-II

Design process and Design Quality

→ Design is represented at a high level of abstraction. level that can be directly traced to specific system objective and more detailed data, functional, behavioral requirements.

→ Quality of the design is assessed with a series of formal technical reviews or design walkthroughs

Characteristics of good design

(1) Design must implement all explicit requirements contained in the analysis model.

(2) Design must be a readable, understandable for generating code, test to support the sw.

(3) Design should provide the complete picture of the sw ie data, functional, behavior of the domain

Quality guidelines

(1) A design should exhibit an architecture.
(a) created using recognizable architectural styles or patterns
(b) composed of components that exhibit good design characteristics
(c) can be implemented in an evolutionary fashion, ie implementation & testing.

(2) A design should be logically partitioned into elements or subsystems (2)

(3) A design should contain distinct representation of data, architecture, interfaces & components.

(4) A design should lead to data structures which are appropriate for the classes to implement & recognizable data patterns.

(5) A design should lead components to exhibit independent functional characteristics.

(6) A design should lead to interfaces to reduce the complexity of connections b/w components & with external environment.

(7) A design should be derived for using repeatable method i.e driven from information obtained during software requirement analysis

(8) A design should be represented using a notation that effectively communicates its meaning.

Quality attributes :- "FURPS" :- Functionality, usability, reliability, performance, supportability.

→ FURPS represents the quality attributes for all s/w design.

(1) **Functionality**
- ↳ Evaluating the features set & capabilities of program.
- ↳ functionality that are delivered.
- ↳ security of the overall system.

(2) **Usability:-**
- ↳ Overall aesthetics, consistency, documentation.

(3) **Reliability:-**
- ↳ measuring the frequency & severity of failure,
- ↳ accuracy of o/p results.
- ↳ mean-time-to failure (MTTF)
- ↳ ability to recover from failure.
- ↳ predictability of the program.

(4) **Performance:-**
- ↳ measured by processing speed, response time, resource consumption, throughput & efficiency.

(5) **Supportability:-**
- → Combines the ability to extend the program extensibility, adaptability, serviceability, testability, compatibility, configurability

Maintainability:-

Design Concepts:- Fundamental design concepts provide the necessary framework for **"getting it right"**

(1) Abstraction (2) Architecture (3) patterns (4) Modularity (5) Information hiding (6) Functional independence (7) Refinement (8) Refactoring (9) Design classes

Design Concepts:-

(1) **Abstraction:-** Many levels of abstraction can be posed.

(i) Highest level of abstraction, a solution is stated in broad terms using the language of the problem environment

(ii) Lower level of abstraction, a more detailed description of the solution is provided.

(iii) Procedural abstraction, refers to a sequence of instructions that have a specific and limited functions.

Eg:- open for a door contains walk to the door, reach out & grasp knob, turn knob and pull door, step away from moving the door etc.

(iv) Data abstraction is a named collection of data that describes a data object in the context of procedural abstraction eg:- door is data abstraction.

(2) **Architecture :-** The over all structure of the software and the ways in which the structure provides conceptual integrity for a system.

→ Architecture is the structure or organization of program components

→ Different models of architecture.

(i) Structured model represents architecture as an organized collection of program components.

(ii) Framework models increase the level of design abstraction by attempting to identify repeatable architectural design frameworks that are, similar types of applications.

(iii) Dynamic models addren the behavioral aspects of the program architecture, which indicates how structure and system configuration may change as a function of external event occurs.

(iv) Procen models focus on the design of the business or technical procen that the system must accommodate.

(v) functional models can be used to represent the functional hierarchy of a system.

(3) Patterns Design pattern describes a design structure that solves a particular design problem within a specific content and forces that may impact on the manner in which the pattern is applied & used.

→ The design pattern is to provide a description that enables a designer to determine

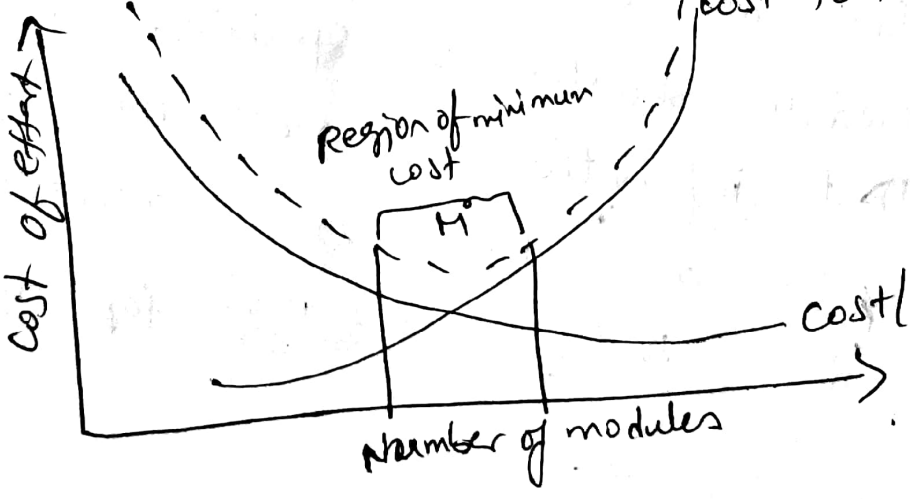
(i) whether the pattern is applicable to the current work.

(ii) whether the pattern can be reused (saving the design time).

(iii) whether the pattern can serves as a guide for developing a similar pattern with different functionality and structure.

(4) Modularity:- Software is divided into separately named and addressable components, sometimes called modules that are integrated to satisfy problem requirements.

→ consider two problems P1 and P2. If the perceived complexity of P1 is greater than the perceived complexity of P2. i.e effort required to solve P1 is greater than the effort required to solve P2.



→ From the above figure the effort of cost to develop an individual software module does decreases as the total number of module increases.

→ As the no. of modules grows, the effort cost anociated with integrating the modules also grows.

→ under modularity and over modularity should be avoided.

→ The modularizing of a design is done so that development can be more easily planned.

→ software increments can be defined and delivered

→ changes can be more easily accomodated,

testing and debugging can be conducted more efficiently, long term maintenance can be conducted without serious side effects.

(5) Information Hiding:- suggests that modules can be characterized by design decisions that each hides from all others.

→ modules should be specified and designed so that information i/e algorithms and data contained within a module is inaccessible to other modules that have no need for such information.

→ Abstraction helps to define the procedural entities that make up the software.

→ Hiding defines and enforces access constraints to both procedural detail with in a module and any local data structure used by the module.

→ The use of information hiding as a design criterion for modular systems provides the greatest benefits when modifications are required during testing.

(6) Functional Independence :- The concept of functional independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding. [Functional independence FOI)

→ FOI is achieved by developing modules with "single minded" function and an "aversion" to excessive interaction with other modules.

→ Independence is assessed using two qualitative criteria

(i) Cohesion :- is a natural extension of the information hiding concept. It is an indication of the relative functional strength of a module.

(ii) Coupling :- It is an indication of the relative interdependence among modules. It is an interconnection among modules in a s/w structure.

→ It depends on the interface complexity b/w modules, the point at which entry or reference is made to a module, and what data pass across the interface.

(7) **Refinement:-** It is top-down design strategy originally proposed by Niklaus Wirth.

→ It is a process of elaboration

→ It begins with a statement of functions or description of data that is defined at a high level of abstraction.

→ The statement describes function or information conceptually but provides no information about the internal workings of the function or the internal structure of the data.

→ Refinement causes the designer to elaborate on the original statement, providing more and more details as each successive refinement occurs

→ Refinement helps the designer to reveal a low level details as design progresses.

(8) **Refactoring:-** Refactoring is a reorganization technique that simplifies the design or code of a component without changing its function or behavior

→ Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code.

→ When s/w is refactored, the existing design is examined for redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures or any other design failure that can be corrected to a better design.

(9) Design classes As design model evolves the s/w team must define a set of design classes

① refine the analysis classes by providing design detail that will enable the classes to be implemented.

② create a new set of design classes that implement a s/w infrastructure to support the business solution.

There are 5 different types of design classes.

(i) User interface classes:- defines all abstractions that are necessary for human computer interaction (HCI) Eg:- checkbox, an order form, for machine.

(ii) Business domain classes :- Classes identify the attributes and services (methods) that are required to implement some element of the business domain.

(iii) Process classes :- implement lower-level business abstractions required to fully manage the business domain classes.

(iv) Persistent classes :- represent data stores i.e database that will persist beyond the execution of the software.

(v) System classes :- Implement s/w management and control functions that enable the system to operate & communicate within its computing environment and within the outside world.

Four characteristics of a well-formed design class :-

(1) complete and sufficient :- A design class should be the complete encapsulation of all attributes and methods that can reasonably be expected to exist for the class.

→ sufficiency ensures that the design class contains only those methods that are sufficient to achieve the intent of the class, no more and no less.

② **Primitiveness** :- Methods associated with a design class should be focused on accomplishing one service for the class.

→ once the service has been implemented with a method, the class should not provide another way to accomplish the same thing.

Eg:- videoclip [setstartpoint()
setendpoint()

③ **High cohesion**:- A cohesive design class has a small, focused set of responsibilities and single mindedly applies attributes and methods to implement those responsibilities. Eg: videoclip → video editing slw may contain a set of methods for editing the video clip.

④ **Low coupling**:- Within the design model, it is necessary for design classes to collaborate with one another, → collaboration should be kept to an acceptable minimum.

→ If the design model is highly coupled the system is difficult to implement, to test, to maintain over time.

→ Design classes within a subsystem should have only limited knowledge of classes in other subsystems.

→ This restriction is called law of Demeter suggests that a method should only send menages to methods of neighbouring classes.

The Design Model

→ The design model can be viewed in two different dimensions

① Process dimensions :- indicates the evolution of the design model as design tasks are executed as part of the software process.

② Abstraction dimension :- Represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively.

→ The elements of the design model use many of the same UML diagrams that are used in the analysis model.

→ The digrams (UML) are refined and elaborated as a part of design.

→ More implementation specific details is provided.

→ More architectural structure & styles, components that reside within the architecture and interfaces b/w the components are described.

(1) Data design elements

→ data design sometimes referred to as data architecting creates a model of data / information that is represented at a high level of abstraction.

→ At the program component level, the design of data structures and the associated algorithms required to manipulate them is essential to the creation of high quality applications.

→ At application level, the translation of a data model into a database is achieving the business objectives of a system.

→ At business level, the collection of information stored in disparate databases and reorganized into a "data warehouse" which enables data mining or knowledge discovery that can have an impact on the sucess of the business itself.

(2) Architectural design Elements :- It gives the overall view of the s/w. It is derived from 3 sources.

(i) Information about application domain for the s/w to be built.

(ii) Specific analysis model elements such as data flow diagrams or analysis classes, their relationships and collaborations for the problem.

(iii) The availability of architectural patterns.

(3) Interface design Elements

→ There are 3 important elements of interface design

(i) The user interface (UI)

(ii) External interfaces to other systems, devices, networks or other producers or consumers of information

(iii) Internal interfaces between various design components.

→ The design of UI incorporates aesthetic elements
Eg: layout, color, graphics, interaction, mechanisms.
i.e UI is a unique subsystems within the overall application architecture.

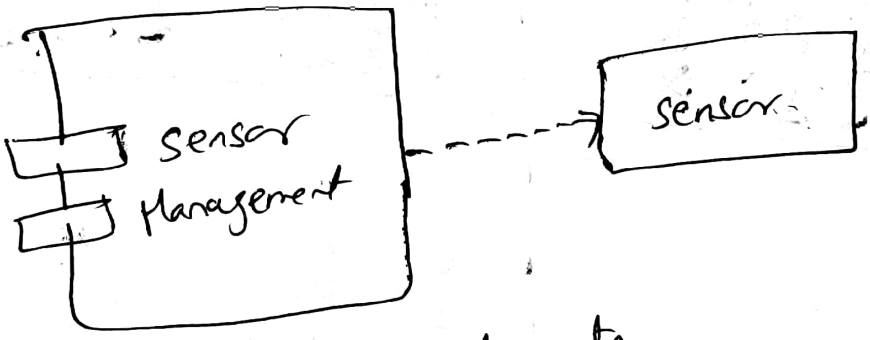
→ (ii) The design of external interfaces requires definitive information about the entity to which information is sent or received.
→ It checks errors, appropriate security features.

(iii) The design of internal interfaces is closely aligned with component level design
i.e analysis of classes, all operations and messages schemes required to enable communication and collaboration blw operations in various classes.

(4) Component-level design elements

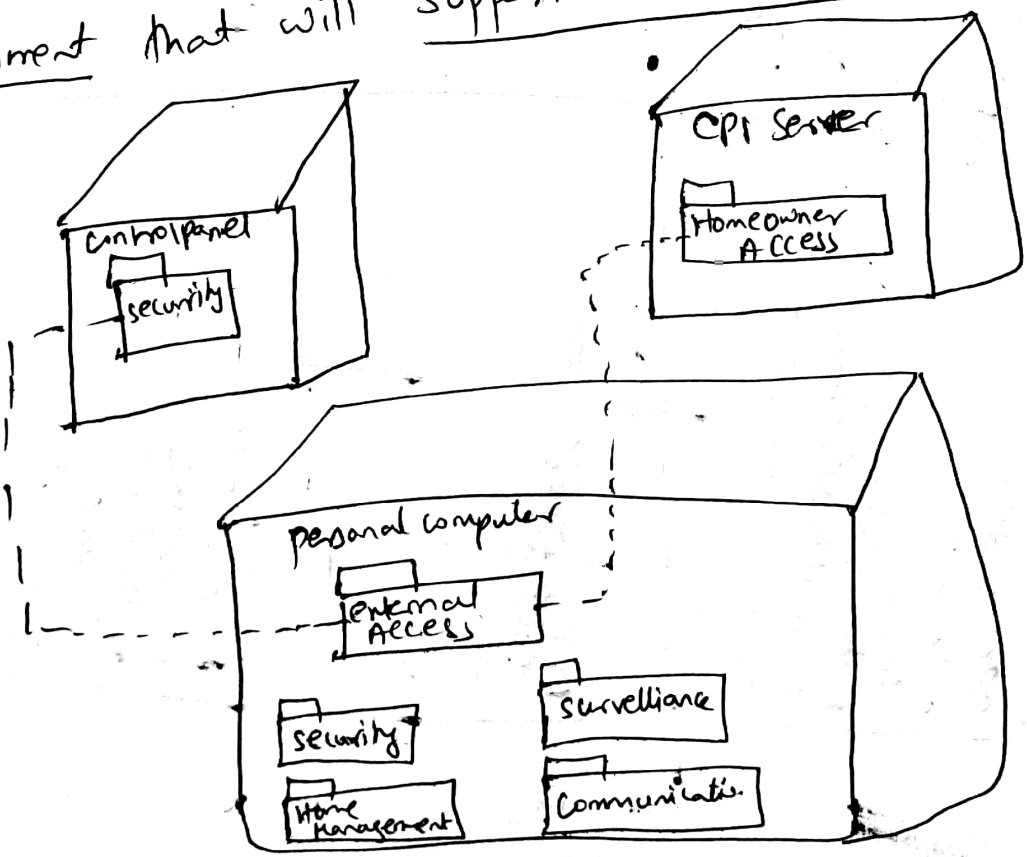
→ Component level design for SW fully describes the internal detail of each SW component.

→ It defines the data structures for all local data objects and algorithmic detail for all processing that occurs within a component and an interface that allows access to all component operations.

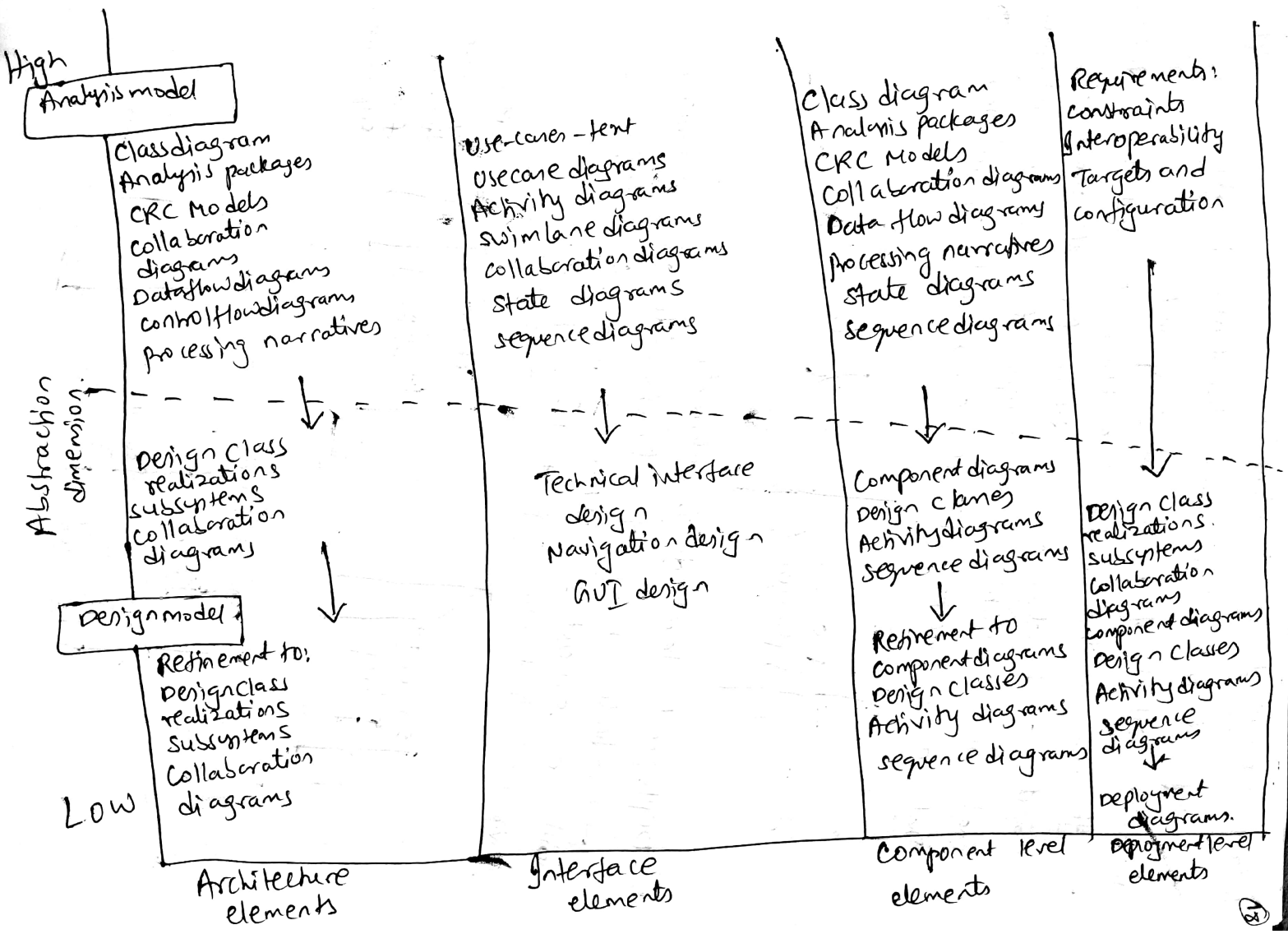


(5) Deployment level design elements

→ Indicates how SW functionality and sub systems will be allocated within physical computing environment that will support the software.



UML deployment diagram for safehome.


High
Analysis model
Class diagram
Analysis packages
CRC Models
Collaboration diagrams
Data flow diagrams
Control flow diagrams
Processing narratives
Use-cases - text
Use case diagrams
Activity diagrams
Swim lane diagrams
Collaboration diagrams
State diagrams
Sequence diagrams
Class diagram
Analysis packages
CRC Models
Collaboration diagrams
Data flow diagrams
Processing narratives
State diagrams
Sequence diagrams
Requirements: constraints
Interoperability
Targets and configuration
Abstraction dimension
Design Class realizations
Subsystems
Collaboration diagrams
Technical interface design
Navigation design
GUI design
Component diagrams
Design classes
Activity diagrams
Sequence diagrams
Refinement to:
Component diagrams
Design classes
Activity diagrams
Sequence diagrams
Design Class realizations
Subsystems
Collaboration diagrams
Component diagrams
Design classes
Activity diagrams
Sequence diagrams
Deployment diagrams.
Design model
Refinement to:
Design class realizations
Subsystems
Collaboration diagrams
Low
Architecture elements
Interface elements
Component level elements
Deployment level elements

Software Architecture

→ What is Architecture?

→ overall shape of the physical structure.

→ It is the manner in which the various components of the building are integrated to form a cohesive whole.

→ It is the visual impact of the building, the way textures, colors, materials combined to create the external facade and internal "living environment"

→ design of lighting fixture, type of flooring; placement of wall hangings and totally it is an "Art".

→ The software architecture of a program or computing system is the structure or structures of the system, which comprise s/w components, externally visible properties of those components and the relationships among them.

→ The architecture is not the operational software.

→ It is the representation that enables a software Engineer (1) Analyze the effectiveness of the design in meetings its stated requirements.

(2) consider architectural alternatives at a stage when making design changes is still relatively easy

(3) reduce the risk associated with construction of the s/w.

→ A software component can be a simple program module or an object-oriented class

→ software architecture considers two levels of design
(i) Data design :- Enables us to represent the data component of the architecture in conventional systems and class definitions in object oriented systems.
(ii) Architectural design :- focus on the representation of the structure of s/w components, their properties and interactions.

Why Architecture is important :-
3 key reasons why s/w architecture is important.
(i) Representations of s/w architecture are an enable for communication b/w all parties (stakeholders) interested in the development of a computer-based system.
(ii) The architecture highlights design decisions that will have profound impact on all software Engineering work that follows.
(iii) Architecture "constitutes" a relatively small, intellectually graspable model of how the system is structured and how its components work together.

Data design

→ The data design action translates data objects defined as part of the analysis model into data structures at the software component level and when necessary a database architecture at the application level.

Data design at the Architecture level

→ The challenge is to extract useful information from this data environment, particularly when the information desired is cross functional.

→ To solve this challenge IT community has developed data mining techniques also called knowledge discovery in databases KDD

→ It navigates through existing databases in an attempt to extract appropriate business level information.

→ A data warehouse is a separate data environment that is not directly integrated with day to day application but encompasses all data used by business.

→ Datawarehouse is a large, independent database that has access to the data that are stored in databases that serve the set of applications required by a business

Data design at the component level

→ It focuses on the representation of data structures that are directly accessed by one or more software components.

→ A set of principles are adapted for data specifications

(1) The systematic analysis principles applied to function and behavior should also be applied to data.

i/e → data flow representation,

→ content should also be developed and reviewed.

→ data objects should be identified

→ Alternative data organizations should be considered

→ Impact of data modeling on software design should be evaluated.

(2) All data structures and the operations to be performed on each should be identified.

→ design of an efficient data structures must take the operations to be performed on the data structure.

(3) A mechanism for defining the content of each data object should be established and used to define both data and the

operations applied to it.

→ Class diagrams defines the data items contained within a class and the operations applied to these data items.

(4) Low-level data design decisions should be deffered until late in the design process.

→ A process of stepwise refinement may be used for the design of data.

→ Overall data organization may be defined during requirements analysis,

→ refined during data design work

(5) The representation of a data structure should be known only to those modules that must make direct use of the data contained within structure

→ The concept of information hidding and concept of coupling which improves the quality of software design

(6) A library of useful data structures and the operations that may be applied to them should be developed.

→ A class library data.

(7) A software design and programming language should support the specification and realization of abstract data types

→ implementation of data structures can be made exceedingly difficult if no means for direct specification of the structure exists in the programming language chosen for implementation.

The above principles are basis for component level data design approach. which are integrated in both analysis and design activities.

Architectural Styles and Patterns

→ The software that is built for computer based systems must exhibit one of many architectural styles.

(1) A set of components eg:- database, computational modules

(2) a set of connectors that enable "communication, coordination, & cooperation"

(3) constraints that define how components can be integrated to form the system.

(4) semantic models that enables a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

→ An architecture style is a transformation that is imposed on the design of an entire system.

→ An architecture pattern like an architecture style imposes a transformation on the design of an architecture.

→ Pattern differs from style from no. of ways.

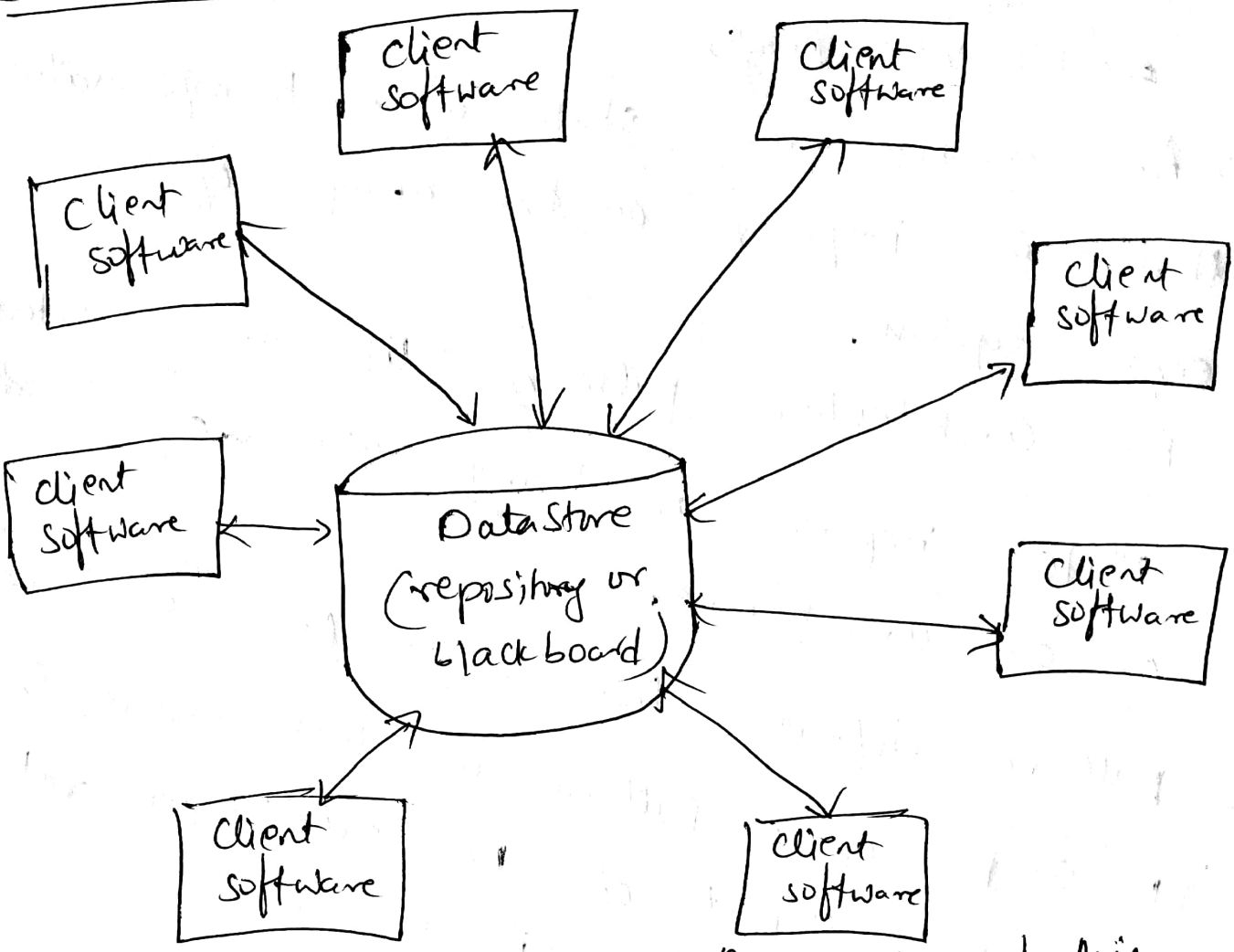
(i) scope of a pattern is less broad, focusing on one aspect of the architecture rather than the architecture in its entirety.

(ii) a pattern imposes a rule on the architecture, describing how the slw will handle some aspect of its functionality at the infrastructure level

(iii) Architectural pattern tend to address specific behavioral issues within the content of the architectural eg:- how real-time application handles synchronization or interrupts.

A Brief - Taxonomy of Architectural styles

Data-centered architecture



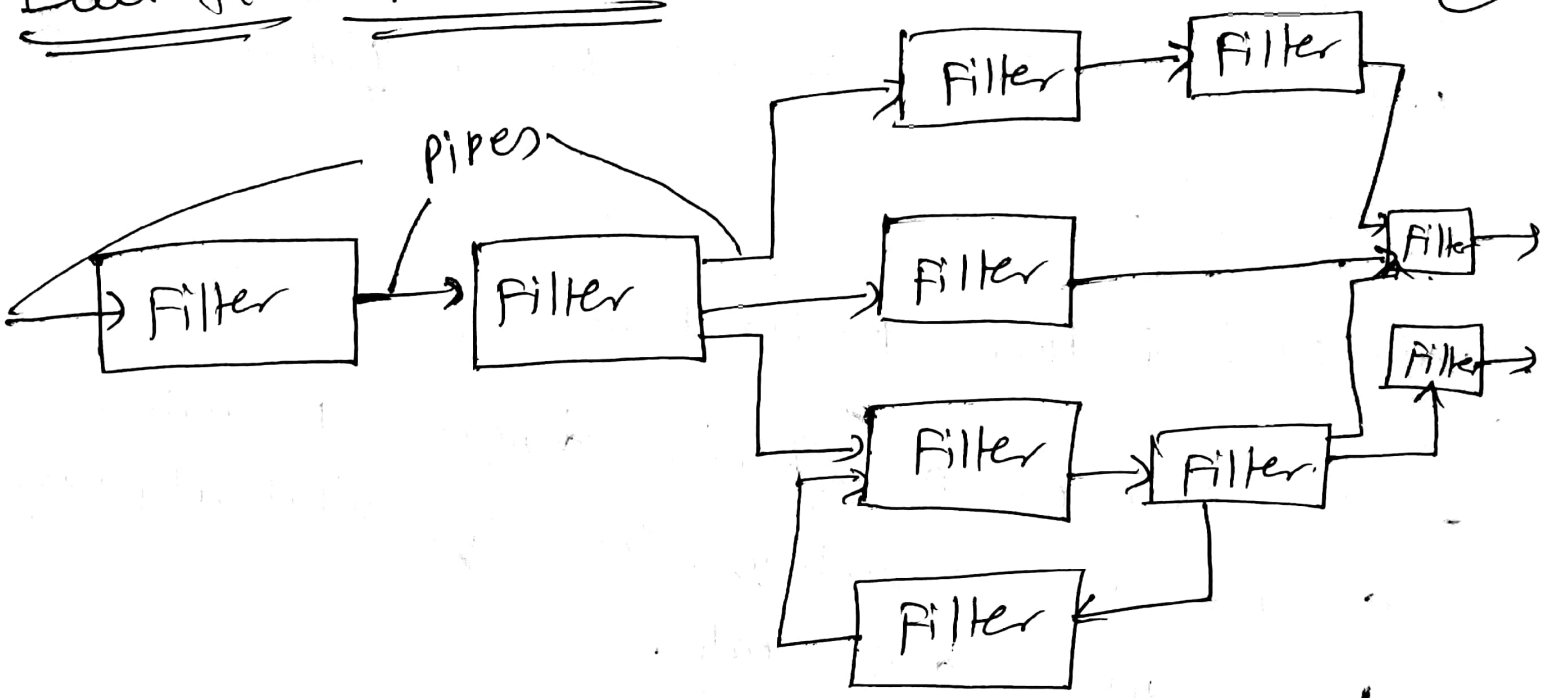
→ A data store resides at the center of this architecture and is accessed frequently by other components that update, add, delete.

→ Existing components can be changed and new client components added to the architecture without concern about their clients.

→ Data can be passed among clients using the blackboard mechanism i.e it coordinates the transfer of information b/w clients.

→ Client components independently execute processes.

Data-flow Architecture



Pipes of Filters

→ It is applied when input data are to be transformed through a series of computational or manipulative components into output data.

→ A Pipe and filter structure has a set of components called filters connected by pipes that transmit data from one component to the next.

→ Each filter works independently of those components upstream and downstream is designed to expect data input of a certain form, and produces data output of a specified form.

→ Filter does not require knowledge of the working of its neighbouring filters

→ If the data flow degenerates into a single line of transforms it is termed batch sequential.

Call and Return Architecture

→ It enables a software designer to achieve a program structure that is relatively easy to modify and scale.

→ Two substyles exists

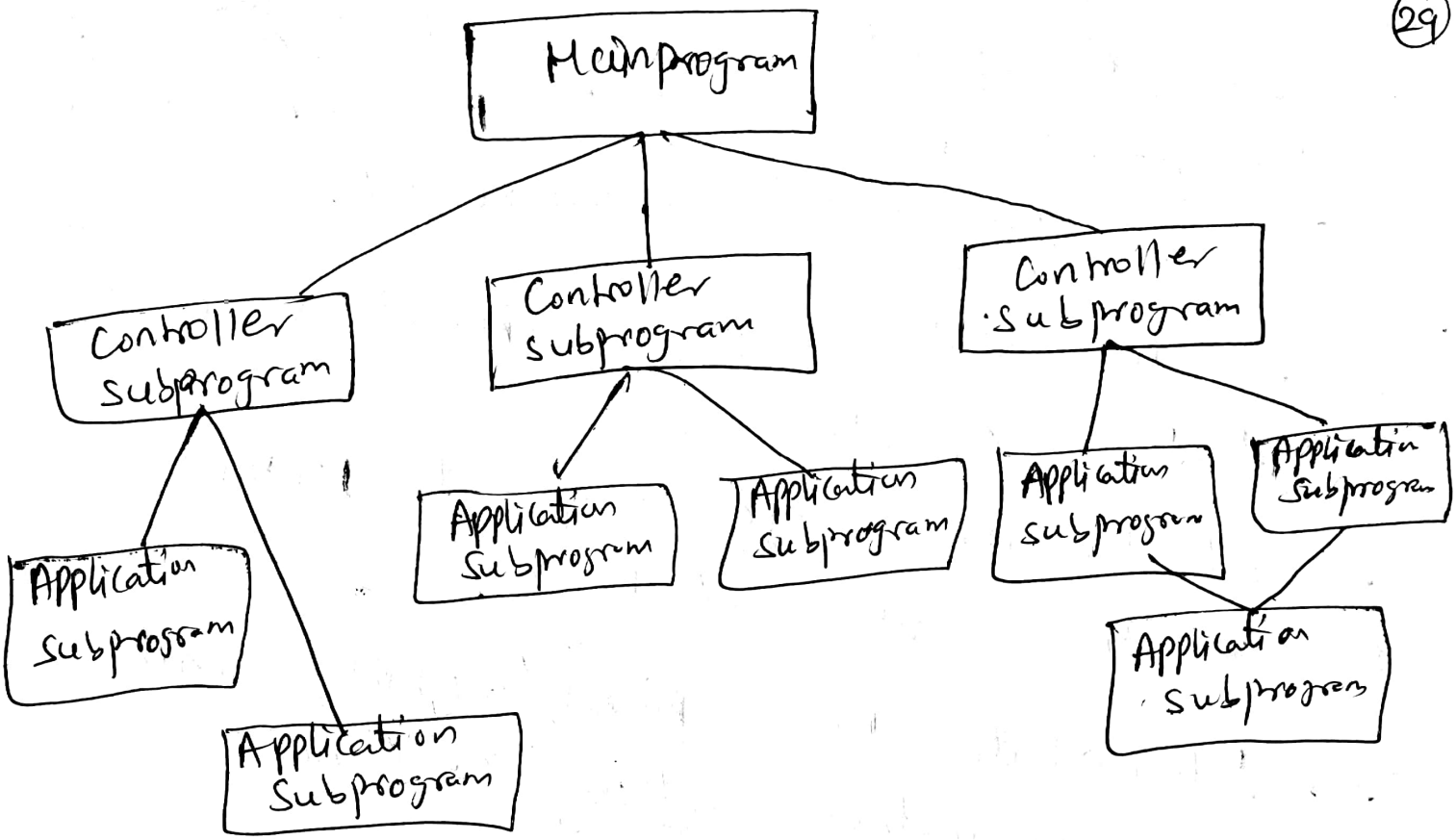
(i) Main program / subprogram architecture

→ This classic program structure decomposes function into a control hierarchy where a main program invokes a no. of program components.

→ which in turn may invoke still other components.

(ii) Remote procedure call architecture

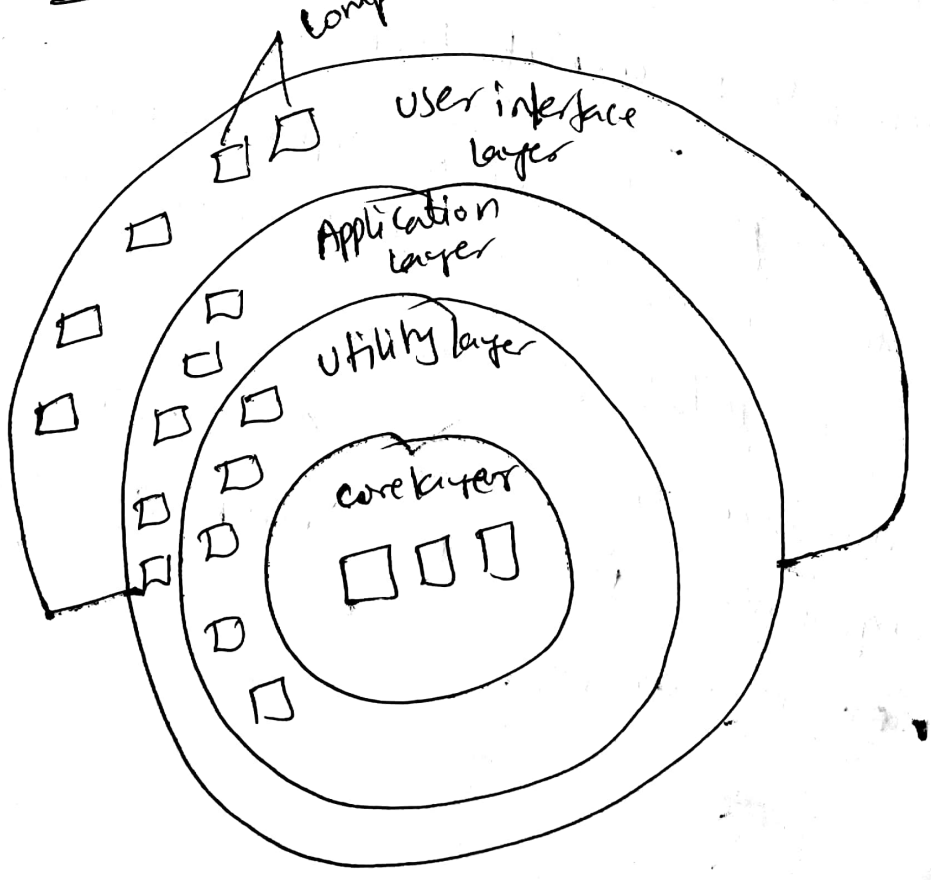
→ The components of main program/ subgram architecture are distributed across multiple computers on a network.



call and return Architecture.

Layered Architecture

→



→ A no. 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 system interfacing.

→ Intermediate layer provides utility services and application software functions.

→ layered style can be combined with a data centered architecture in many database applications.

Object oriented architecture

→ The components of a system encapsulate data and the operations that must be applied to manipulate the data.

→ communication and coordination b/w components is accomplished through message passing.

Architectural Patterns

→ Architectural patterns are bit different

Eg:- Kitchen pattern defines the need for placement of basic kitchen appliances, the need for a sink, the need for cabinets,

→ Architectural patterns for software define a specific approach for handling some behavioral characteristic of the system.

(1) concurrency:- Many applications must handle multiple tasks in a manner that simulates parallelism.

Eg:- operating system management pattern built in OS features that allow components to execute concurrently. i.e processes, scheduling, & other capabilities required to achieve concurrency.

Persistence (ability to store something) – persistent data are stored in a database or file and may be read or modified by other processes at later time.

i.e attributes of a object,
state of the object
stored information for future retrieval & use.

two architectural patterns are used to achieve persistence

(i) database Management system.
(ii) Application level persistence pattern
i.e word processing s/w that manages its own document structure.

Distribution – It addresses the manner in which systems or components within systems communicate with one another in a distributed environment.

→ There are two elements to this problem

(1) The way in which entities connect to one another.

(2) The nature of the communication that occurs.

→ The most common architectural pattern established to address the distribution problem is the broker pattern.

→ The broker acts as a "middle man" blw client component and a server component.

Organization and Refinement

The following questions provide insight into the architectural style that has been derived

Control :- (1) How is control managed within the architecture?

(2) Does a distinct control hierarchy exist

(3) What is the role of components within this control hierarchy?

Data:- (1) How are data communicated b/w components?

(2) How do functional components interact with data components

(3) How do data and control interact within the system.

Architectural Design

→ The s/w to be developed must be put into context i.e the design should define the external entities that the s/w interacts with and the nature of the interaction.

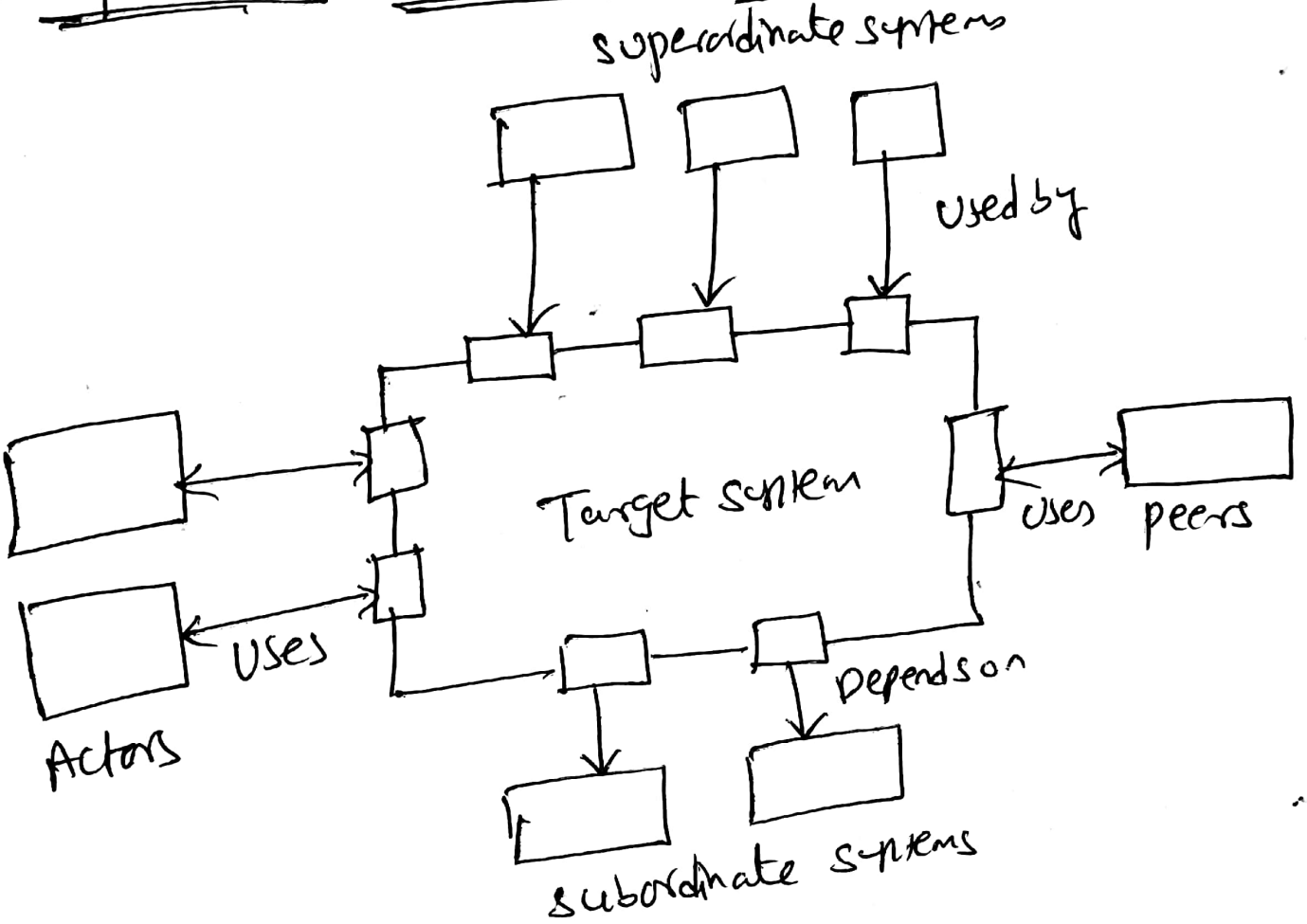
(1) Representing the system in Context

(2) Defining Archetypes → typical example of a certain person or thing

(3) Refining the Architecture into components

(4) Describing instantiations of the system.

5) Representing the system in Context



→ systems that interoperate with the target systems are presented as

(i) Superordinate systems :- Those systems that use the target systems as part of some higher level processing scheme

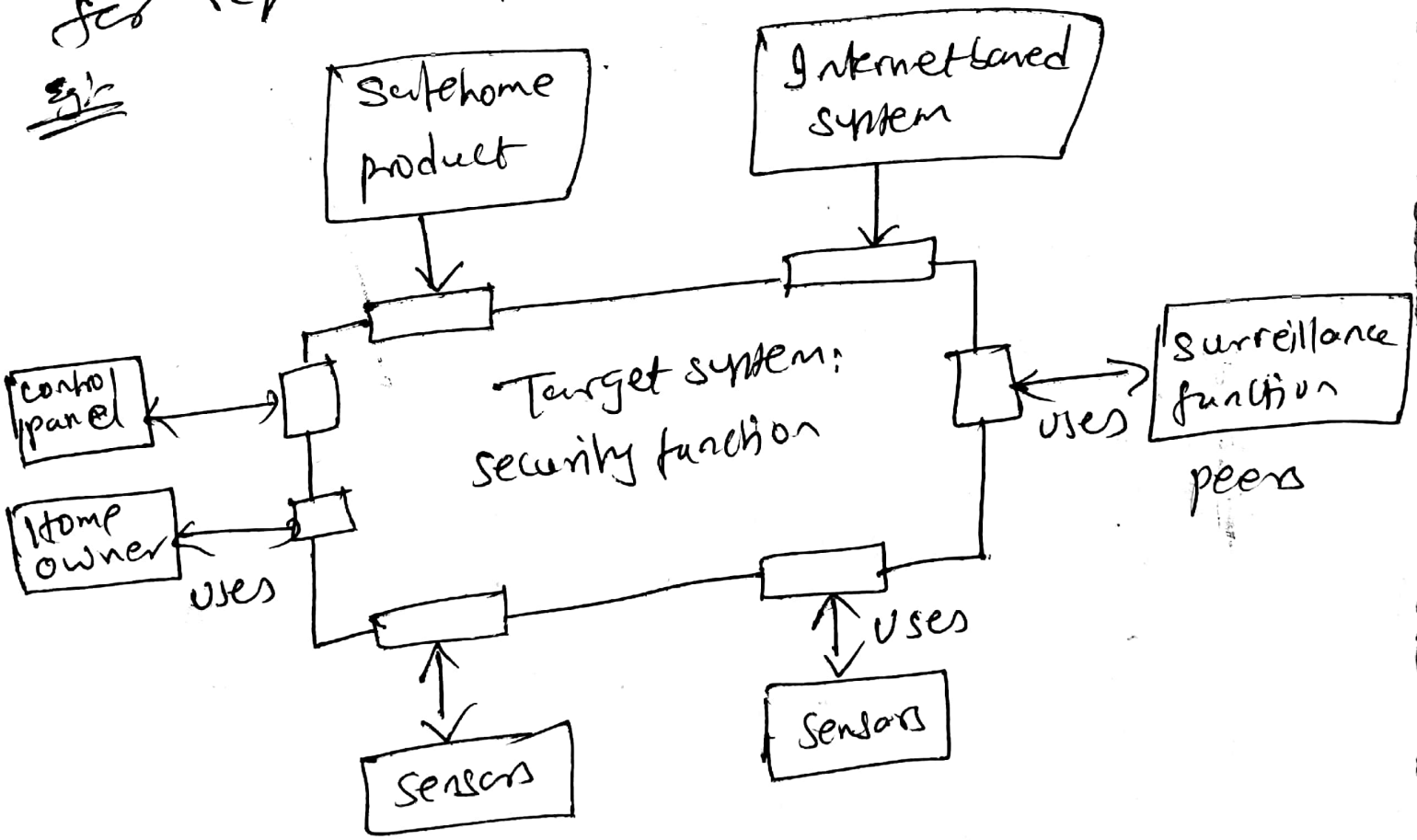
(ii) subordinate systems :- Those systems that are used by the target system and

provide data or processing that are necessary to complete target system functionality.

Peer-level systems:- Those systems that interact on peer-to-peer basis.

Actors:- those entities (people, devices) that interact with the target system by producing or consuming information that is necessary for requisite processing.

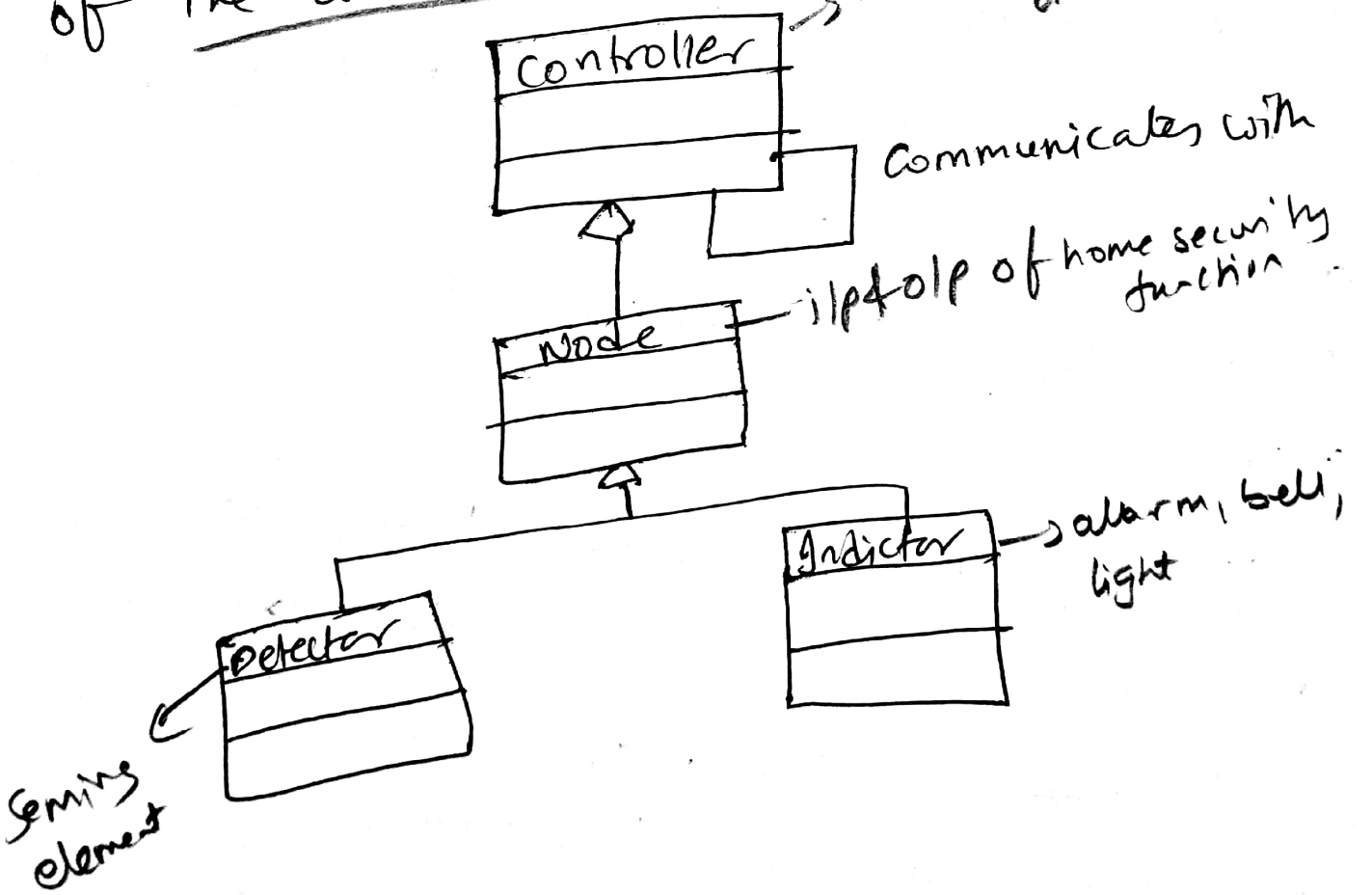
Eg:-



Defining Archetypes

→ An archetype is a class or pattern that represents a core abstraction that is critical to the design of an architecture for the target system.

→ Archetypes can be derived by examining the analysis classes defined as a part of the analysis model.



SafeHome Archetypes are

(i) Node Represents a cohesive collection of input and output elements of the home security function.
→ various sensors
→ a variety of alarm (output) indicators.

(ii) Detector An abstraction that encompasses all sensing equipment that feeds information into the target system.

(iii) Indicator An abstraction that represents all mechanisms (eg:- alarm siren, bell, flashing lights) for indicating that an alarm condition is occuring.

(iv) Controller An abstraction that depicts the mechanism that allows the arming or disarming a node. If controllers reside on a nlw, they have the ability to communicate with one another

(3) Refining the Architecture into components

→ The structure of the system begins to emerge.

→ The architecture must accommodate many infrastructure components that enable application components but have no business connection to the application domain.

→ Eg:- Memory Management components.
Communication components
database components
task management components

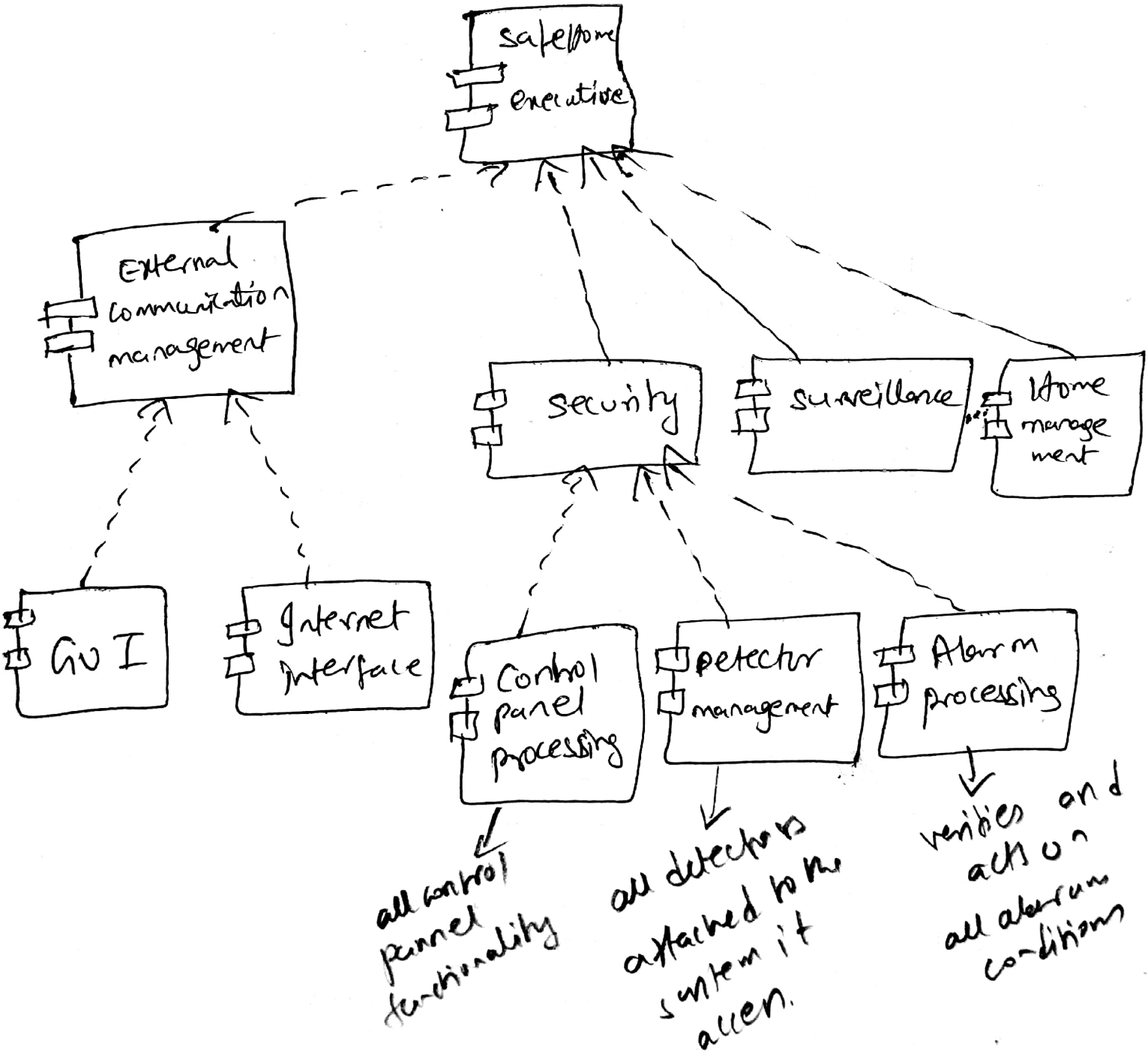
Top level components of safehome security

→ External communication management:- coordinates communication of the security function with external entities Eg:- Internet based systems, external alarm notification.

→ Control panel processing:- manages all control panel functionality.

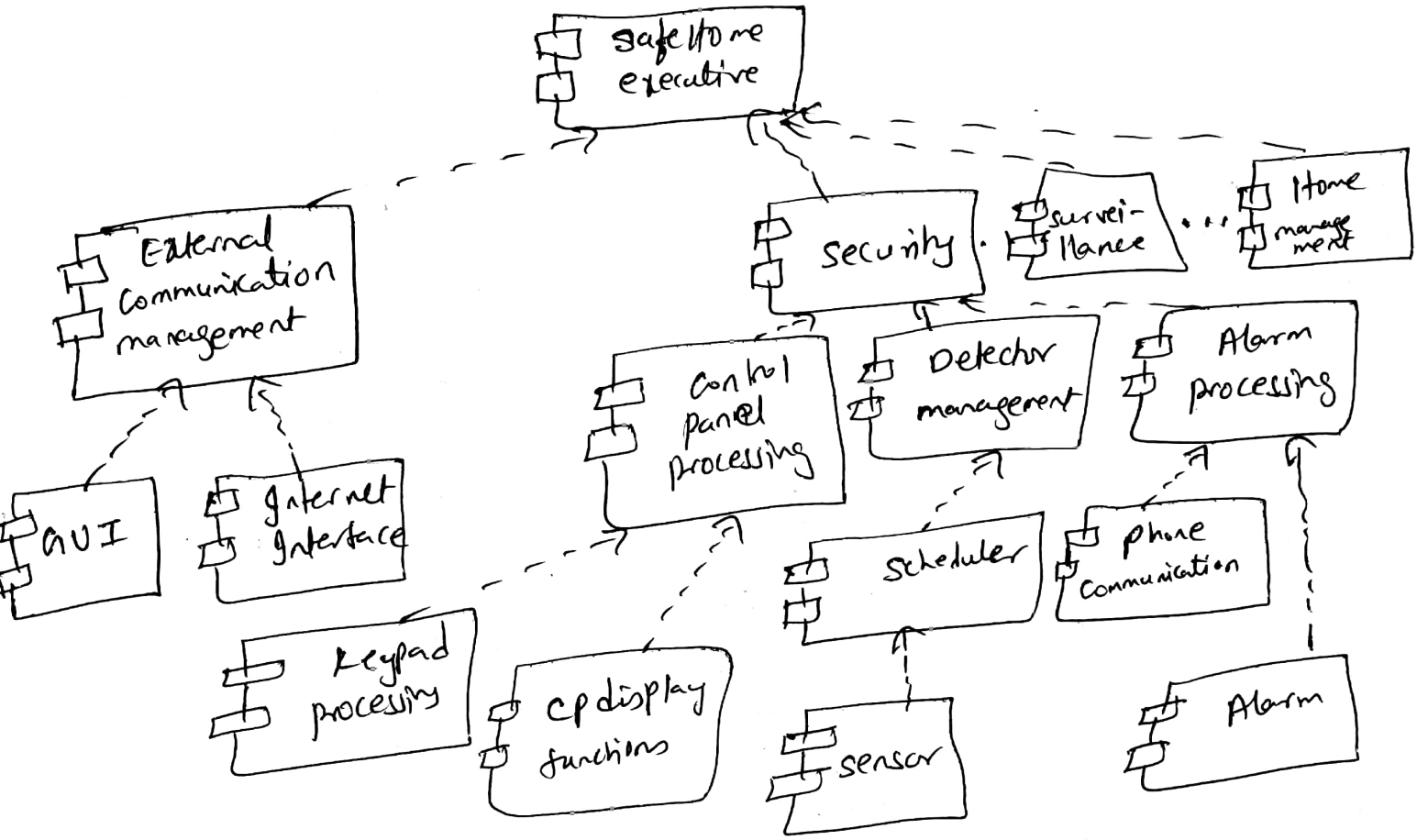
→ Detector management :- Coordinates access to all detectors attached to the system.

→ Alarm processing verifies and acts on all alarm conditions.



Describing Instantiations of the System (4)

→ model is still relatively high level
→ Content of the system has been represented
→ Archetypes that indicate the important abstractions within problem domain have been defined.
→ overall structure of the system is apparent.
→ major s/w components have been identified.
→ An actual instantiation of the architecture is developed.
↳ creation of a real instance
→ Architecture is applied to a specific problem with the intent of demonstrating the structure and components.


safeHome executive
External communication management
security
surveillance
Home management
GUI
Internet Interface
control panel processing
Detector management
Alarm processing
Keypad processing
cp display functions
Scheduler
phone communication
sensor
Alarm