Module-3:: Design Concepts

- The main aim of design engineering is to generate a model which shows firmness, delight and commodity.
- Software design is an iterative process through which requirements are translated into the blueprint for building the software.
- A design is generated using the recognizable architectural styles and compose a good design characteristic of components and it is implemented in evolutionary manner for testing.
- A design of the software must be modular i.e the software must be logically partitioned into elements.
- In design, the representation of data , architecture, interface and components should be distinct.
- A design must carry appropriate data structure and recognizable data patterns.
- Design components must show the independent functional characteristic.
- A design creates an interface that reduces the complexity of connections between the components.
- A design must be derived using the repeatable method.

1.What is Software Design? Explain the Design Concepts in detailed ?

> *"Software Design is the process to transform the user requirements into some suitable form, which helps the programmer in software coding and implementation"*

During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document.
The aim of this phase is to transform the SRS document into the design document.

* Different modules required.
* Control relationships among modules.
* Interface among different modules.
* Data structure among the different modules.
* Algorithms required to implement among the individual modules

Objectives of Software Design:

1. **Correctness:**
A good design should be correct i.e. it should correctly implement all the functionalities of the system.
2. Efficiency:
A good software design should address the resources, time, and cost optimization issues.
3. Understandability:
A good design should be easily understandable, for which it should be modular and all the modules are arranged in layers.
4. Completeness:
The design should have all the components like data structures, modules, and external interfaces.
5. Maintainability:
A good software design should be easily amenable to change whenever a change request is made from the customer side.

Quality attributes:

The attributes of design name as 'FURPS' are as follows:

Functionality:
It evaluates the feature set and capabilities of the program.

Usability:
It is accessed by considering the factors such as human factor, overall aesthetics, consistency and documentation.

Reliability:
It is evaluated by measuring parameters like frequency and security of failure, output result accuracy, the mean-time-to-failure(MTTF), recovery from failure and the the program predictability.

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

Supportability:

It combines the ability to extend the program, adaptability, serviceability. These three term defines the maintainability.

Testability, compatibility and configurability are the terms using which a system can be easily installed and found the problem easily.

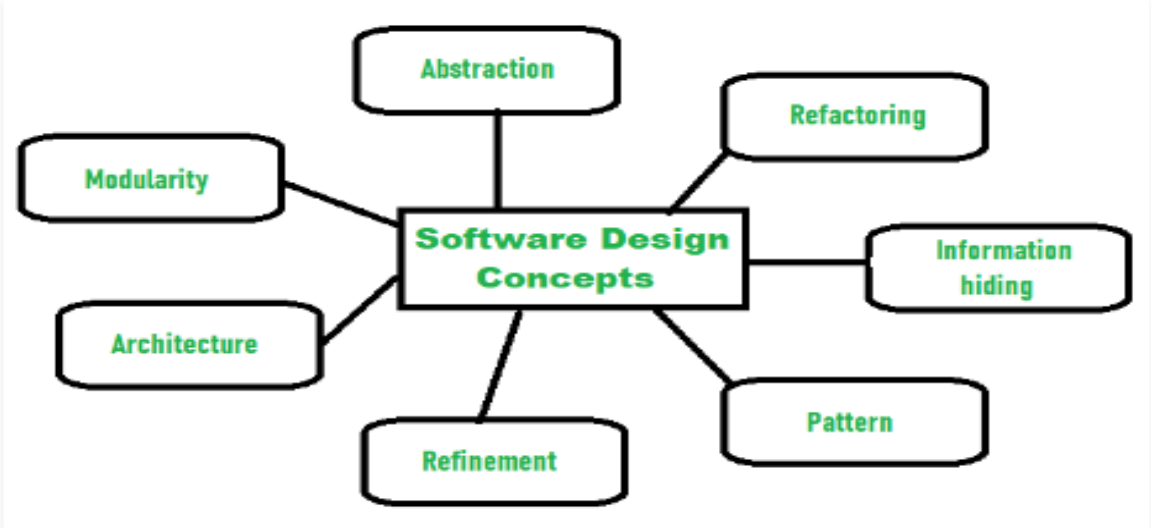
Supportability also consists of more attributes such as compatibility, extensibility, fault tolerance, modularity, reusability, robustness, security, portability, scalability.

Design concepts:

The set of fundamental software design concepts are as follows:

1. Abstraction

- A solution is stated in large terms using the language of the problem environment at the highest level abstraction.
- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- A collection of data that describes a data object is a data abstraction.



2. Architecture

- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

3. Patterns

A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

4. Modularity

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.

5. Information hiding

Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria i.e Cohesion and coupling.

Cohesion

- Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

Coupling is an indication of interconnection between modules in a structure of

software.

7. Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

8. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.

9. Design classes

- The model of software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on features of the problem which are user visible.

2.How the Software is designed by using the Software Engineering process ?

There are three different levels of software design.

1. **Architectural Design:**

The architecture of a system can be viewed as the overall structure of the system & the way in which structure provides conceptual integrity of the system. The architectural design identifies the software as a system with many components interacting with each other.

At this level, the designers get the idea of the proposed Solution domain.

2. Preliminary/high-level design:

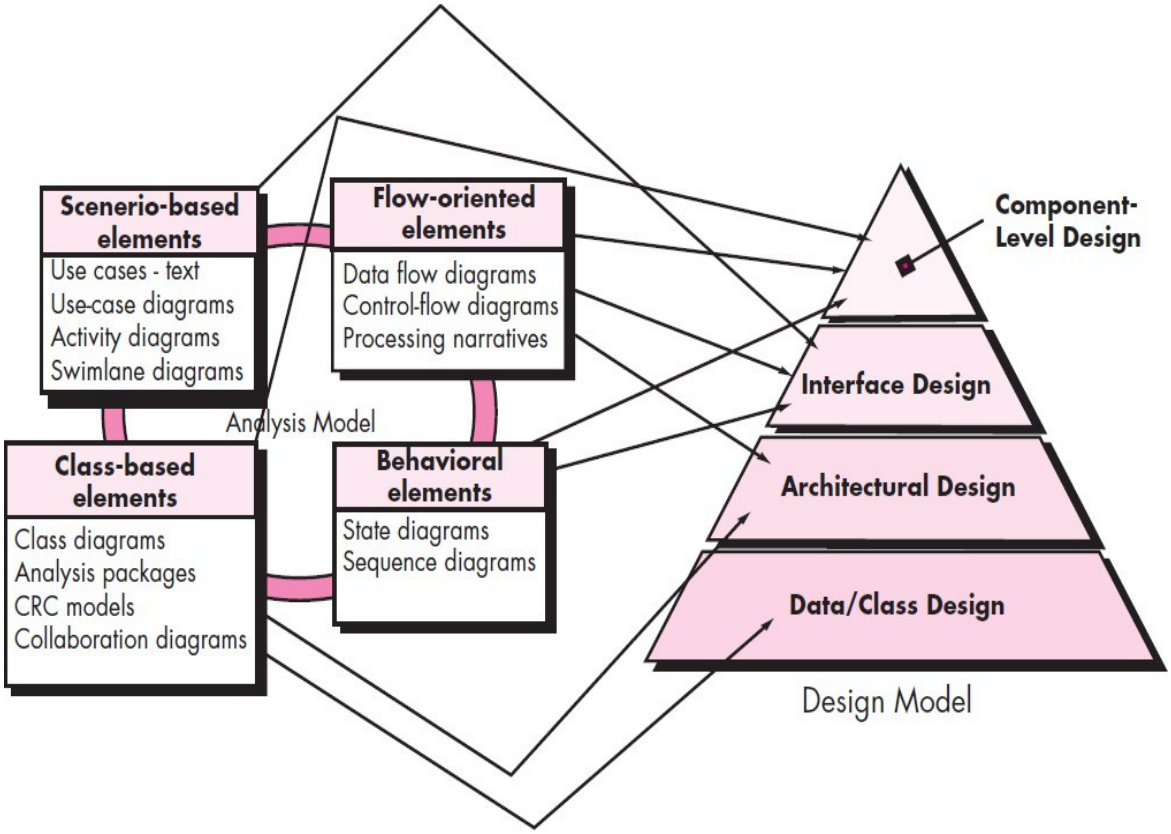
The problem is decomposed into a set of modules, the control relationship among various modules identified, and also the interfaces among various modules are identified. The outcome of this stage is called the program architecture. Design representation techniques used in this stage are structure chart

and UML.

3. Detailed design:

Once the high-level design is complete, a detailed design is undertaken. In detailed design, each module is examined carefully to design the data structure and algorithms. The stage outcome is documented in the form of a module specification document.

Translating the requirements model into the design model:



3.Explain about the design process of a software ?

The Design Process:

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.

It can be divided into 3 levels:

1. Interface Design/high-level design
2. Architectural Design
3. Detailed Design

1. Interface design should include the following details:

Precise description of events in the environment, or messages from agents to which the system must respond.
Precise description of the events or messages that the system must produce.
Specification on the data, and the formats of the data coming into and going out of the system.
Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

2. Issues in architectural design includes:
Gross decomposition of the systems into major components.
Allocation of functional responsibilities to components.
Component Interfaces
Component scaling and performance properties, resource consumption properties, reliability properties
Communication and interaction between components.

3. The detailed design may include:
Decomposition of major system components into program units.
Allocation of functional responsibilities to units.
User interfaces
Unit states and state changes
Data and control interaction between units
Data packaging and implementation, including issues of scope and visibility of program elements
Algorithms and data structures

There are five different types of design classes and each type represents the layer of the design architecture these are as follows:

1. User interface classes

- These classes are designed for Human Computer Interaction(HCI).
- These interface classes define all abstraction which is required for Human Computer Interaction(HCI).

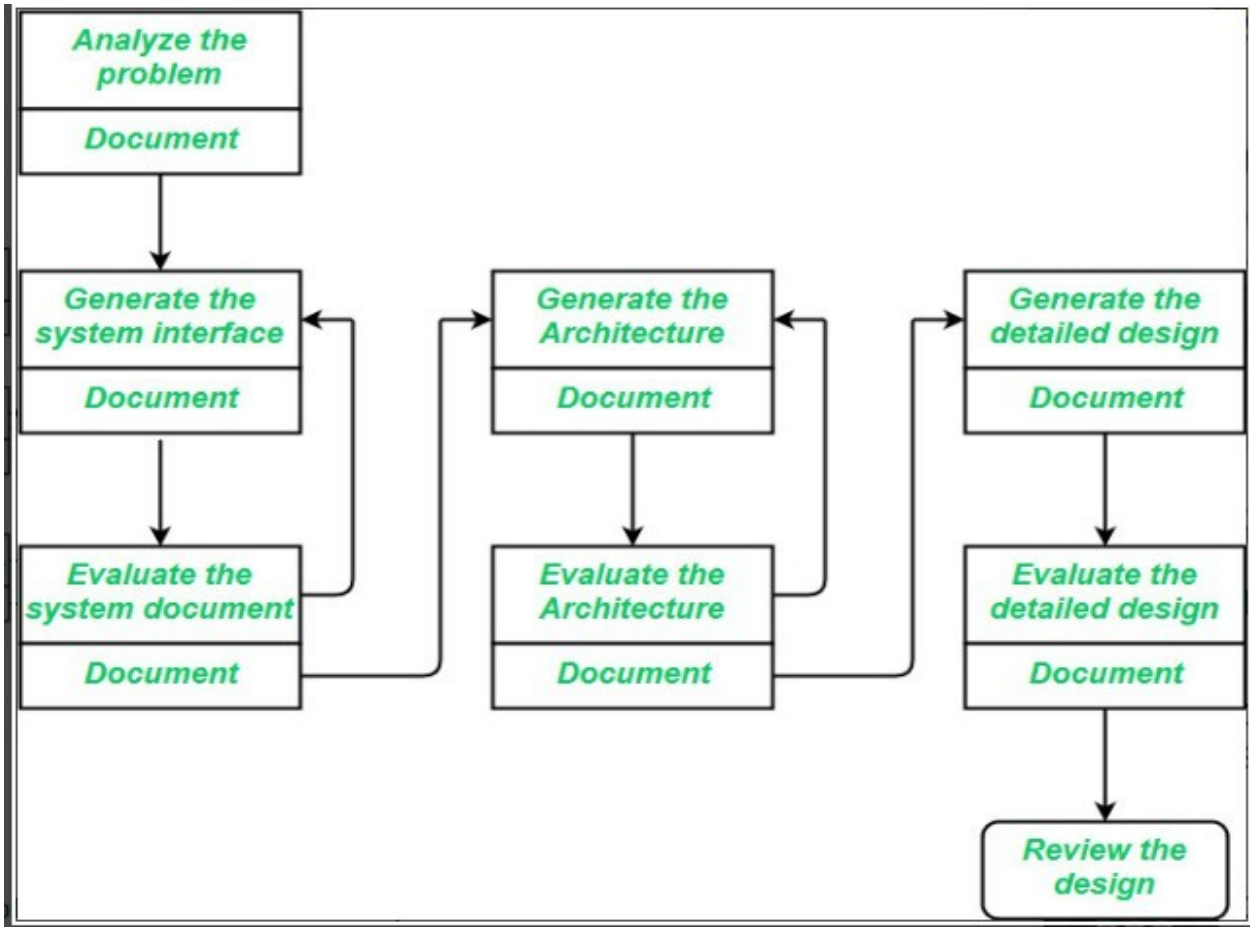
2. Business domain classes

- These classes are commonly refinements of the analysis classes.
- These classes are recognized as attributes and methods which are required to implement the elements of the business domain.

3. Process classes
It implement the lower level business abstraction which is needed to completely manage the business domain class.

4. Persistence classes
It shows data stores that will persist behind the execution of the software.

5. System Classes
System classes implement software management and control functions that allow to operate and communicate in computing environment and outside world.



Software Quality Guidelines and Attributes:
• The design must implement all of the explicit requirements contained in the requirements model, and it must accommodate all of the implicit requirements desired by stakeholders.

• The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
• The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

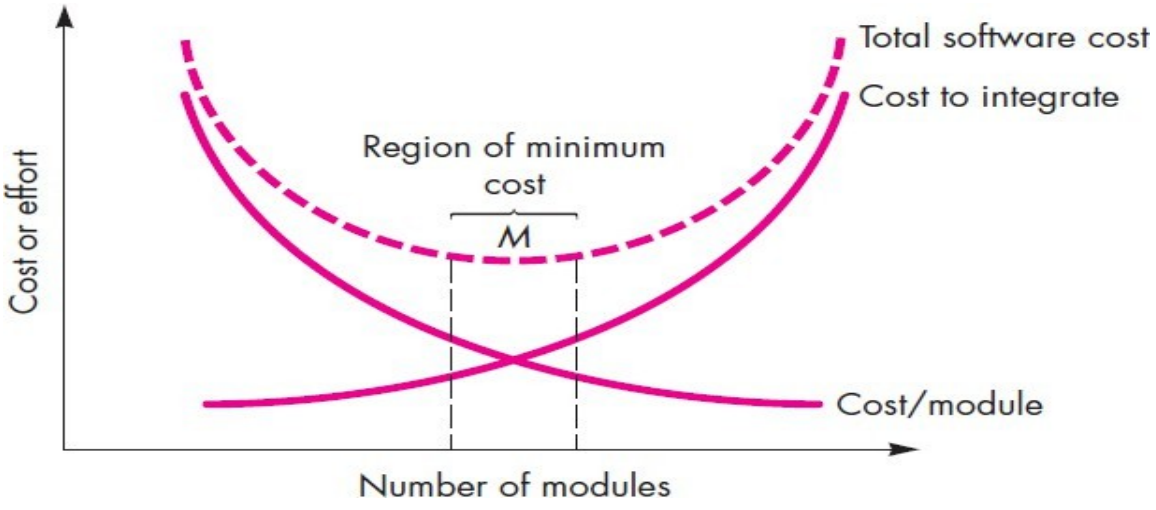
Quality Guidelines:

1. A design should exhibit an architecture that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics (these are discussed later in this chapter), and (3) can be implemented in an evolutionary fashion,[2] thereby facilitating implementation and testing.
2. A design should be modular; that is, the software should be logically partitioned into elements or subsystems.
3. A design should contain distinct representations of data, architecture, interfaces, and components.

4. A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
5. A design should lead to components that exhibit independent functional characteristics.
6. A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
7. A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
8. A design should be represented using a notation that effectively communicates its meaning.

Quality Attributes:

1. Functionality
2. Usability
3. Reliability
4. Performance
5. Supportability

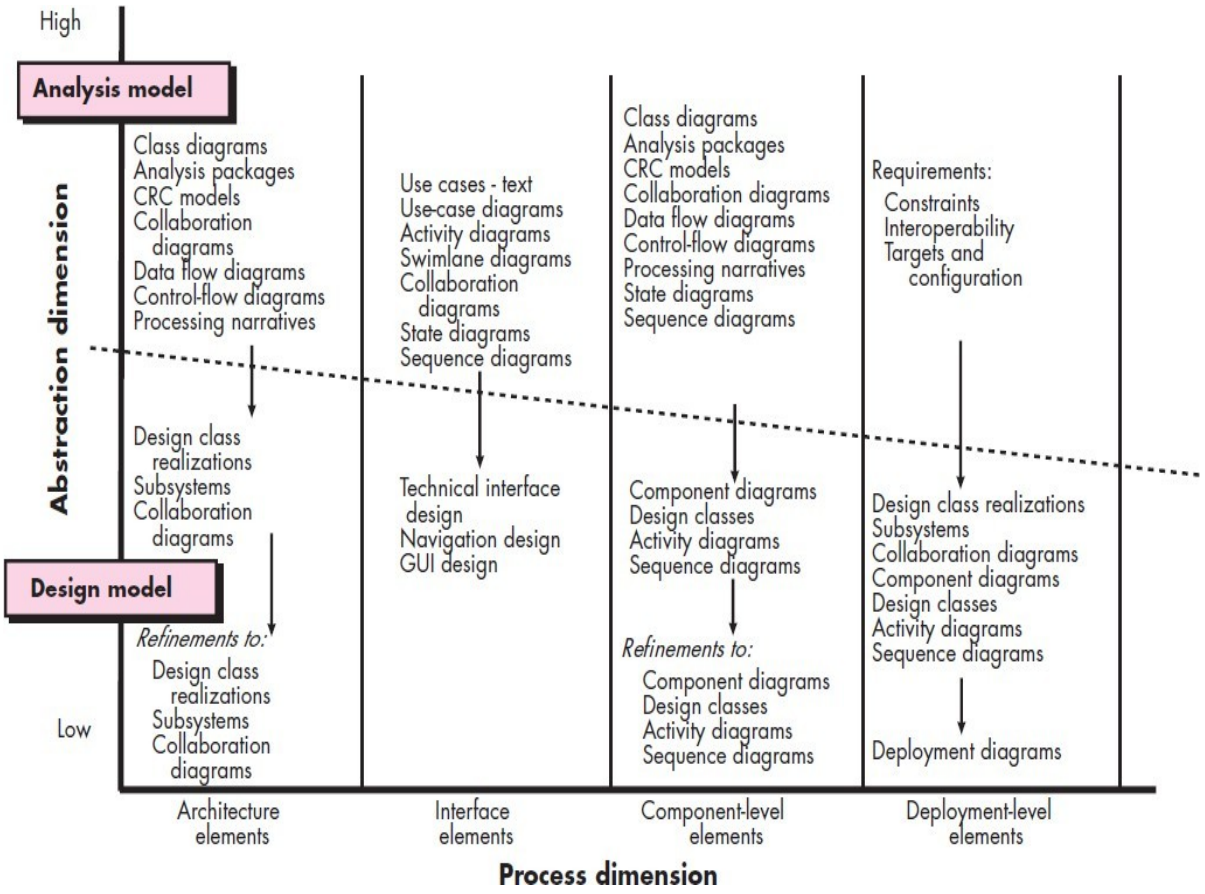


4.How to develop the Software design model ?

The design model:

A design model in software engineering is an object-based picture or pictures that represent the use cases for a system.
Design modeling in software engineering represents the features of the software that helps engineer to develop it effectively, the architecture, the user interface, and the component level in detail.

1. **Data design**
2. **Architectural design**
3. **User Interfaces design**
4. **Component level design**



Architectural Design

The software needs the architectural design to represents the design of software. IEEE defines architectural design as “the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.” The software that is built for computer-based systems can exhibit one of these many architectural styles.

Each style will describe a system category that consists of :

1. A set of functionalities
2. The set of connectors will help in coordination, communication, and cooperation between the components.
3. Conditions that how components can be integrated to form the system.
4. Semantic models that help the designer to understand the overall properties of the system components (eg: a database, computational modules) that will perform a function required by the system.

6.Discuss about various Architecture Styles in detailed ?

Architectural styles:

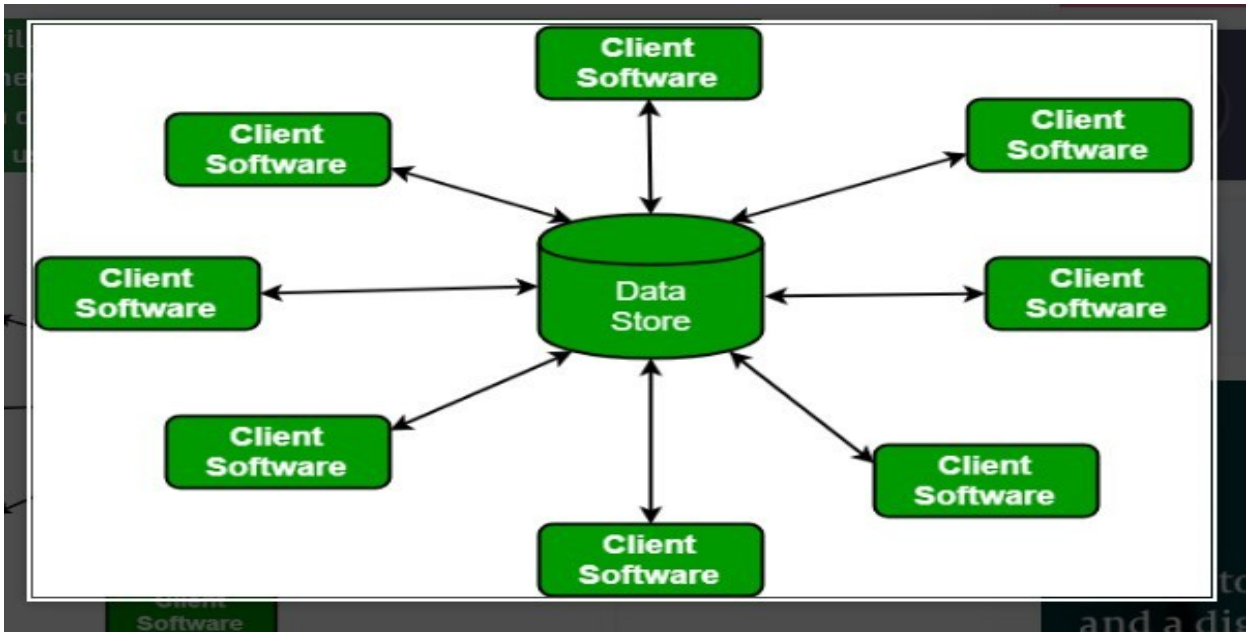
1. Data centered architectures:

A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.

The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.

This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.

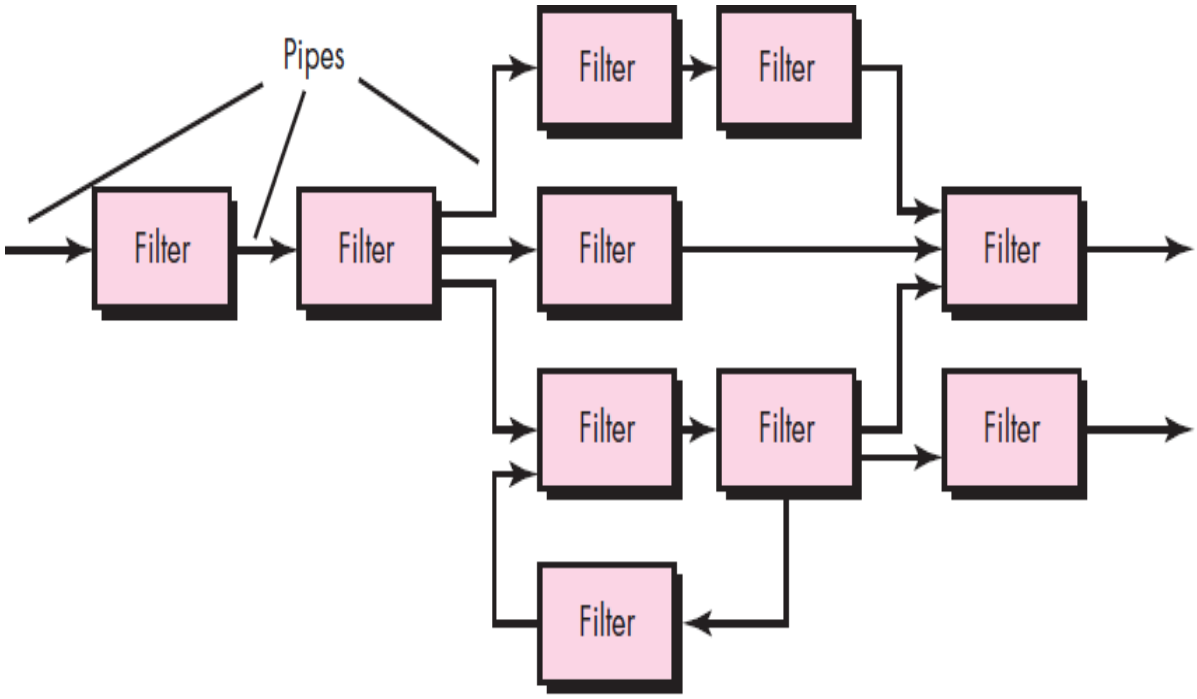
Data can be passed among clients using blackboard mechanism.



2. Data-flow architectures:

This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data. A pipe-and-filter pattern 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 (to the next filter) of a specified form. However, the filter does not require knowledge of the workings of its neighboring filters.



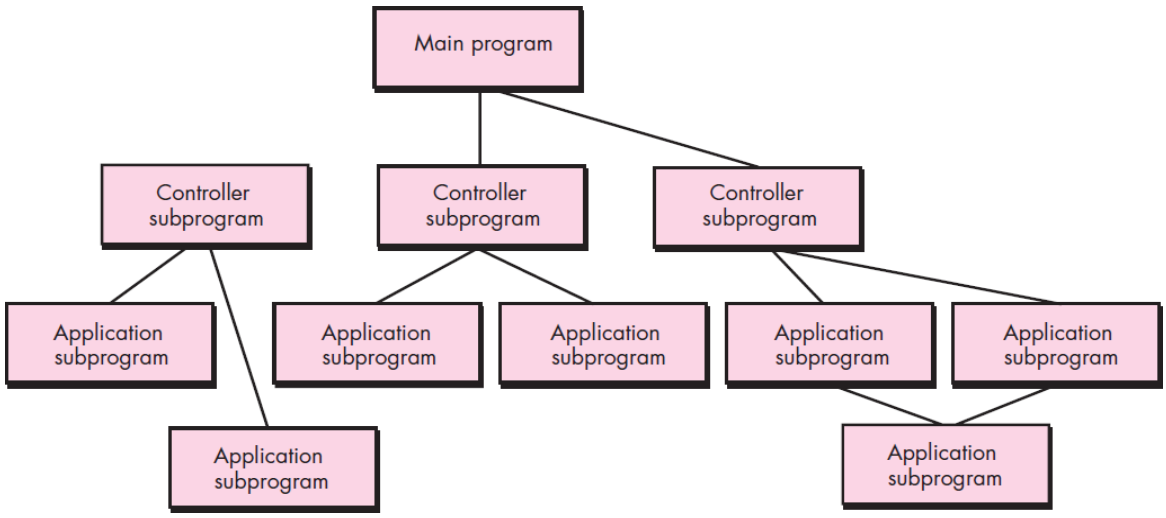
Pipes and filters

3. Call and return architectures:

This architectural style enables you to achieve a program structure that is relatively easy to modify and scale.

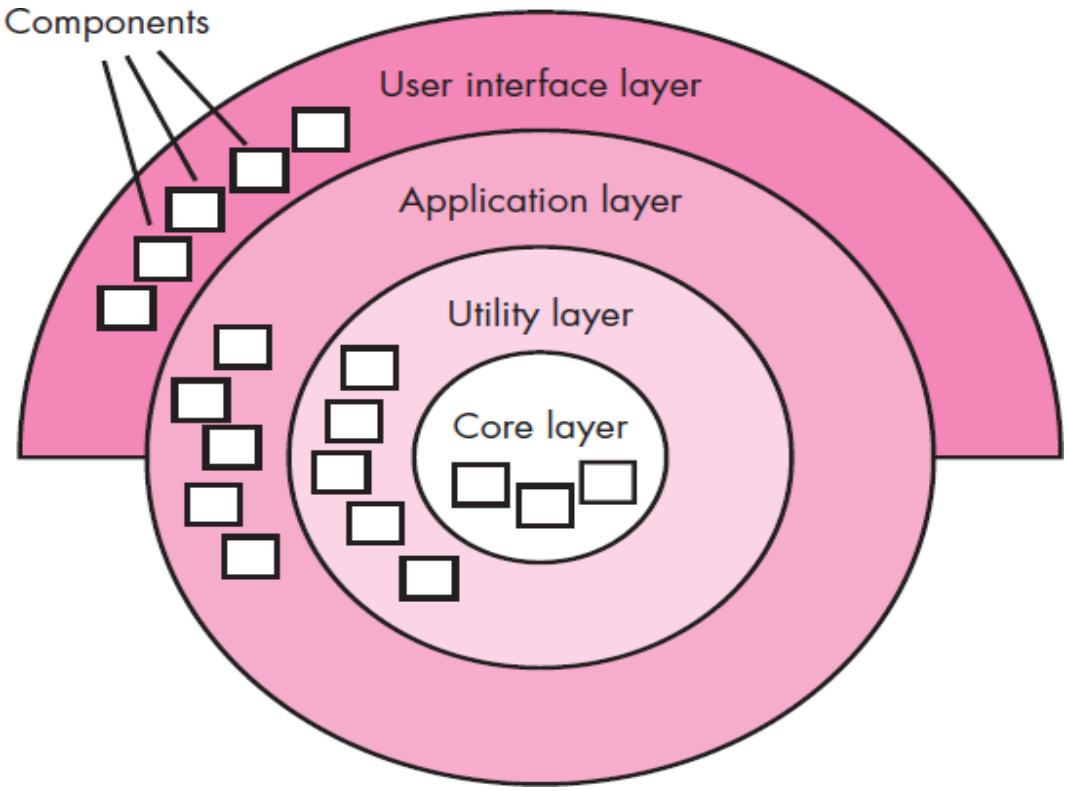
A number of sub styles exist within this category:

a. Main program/subprogram architectures
b. Remote procedure call architecture



Main program/subprogram architecture:

4. Layered architectures. The basic structure of a layered architecture is illustrated 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 system interfacing. Intermediate layers provide utility services and application software functions.



5.Define Software Architecture? Discuss about Architectural Design in detailed

Architectural Design:

1 .It defines an abstraction level at which the designers can specify the functional and performance behavior of the system.

2. It acts as a guideline for enhancing the system (when ever required) by describing those features of the system that can be modified easily without affecting the system integrity.

3. It evaluates all top-level designs.

4. It develops and documents top-level design for the external and internal interfaces.

5. It develops preliminary versions of user documentation.

6. It defines and documents preliminary test requirements and the schedule for software integration.

7. The sources of architectural design are listed below.

8. Information regarding the application domain for the software to be developed

9. Using data-flow diagrams

10. Availability of architectural patterns and architectural styles.

Architectural Design Representation

Architectural design can be represented using the following models.

Structural model: Illustrates architecture as an ordered collection of program components

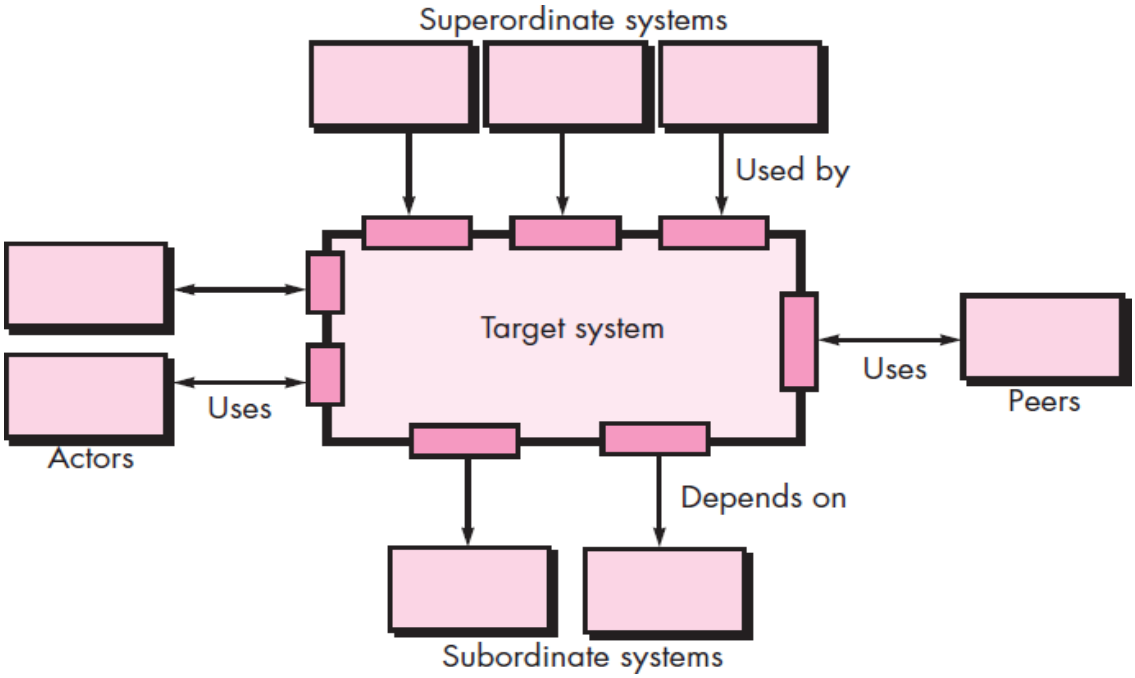
Dynamic model: Specifies the behavioral aspect of the software architecture and indicates how the structure or system configuration changes as the function changes due to change in the external environment

Process model: Focuses on the design of the business or technical process, which must be implemented in the system

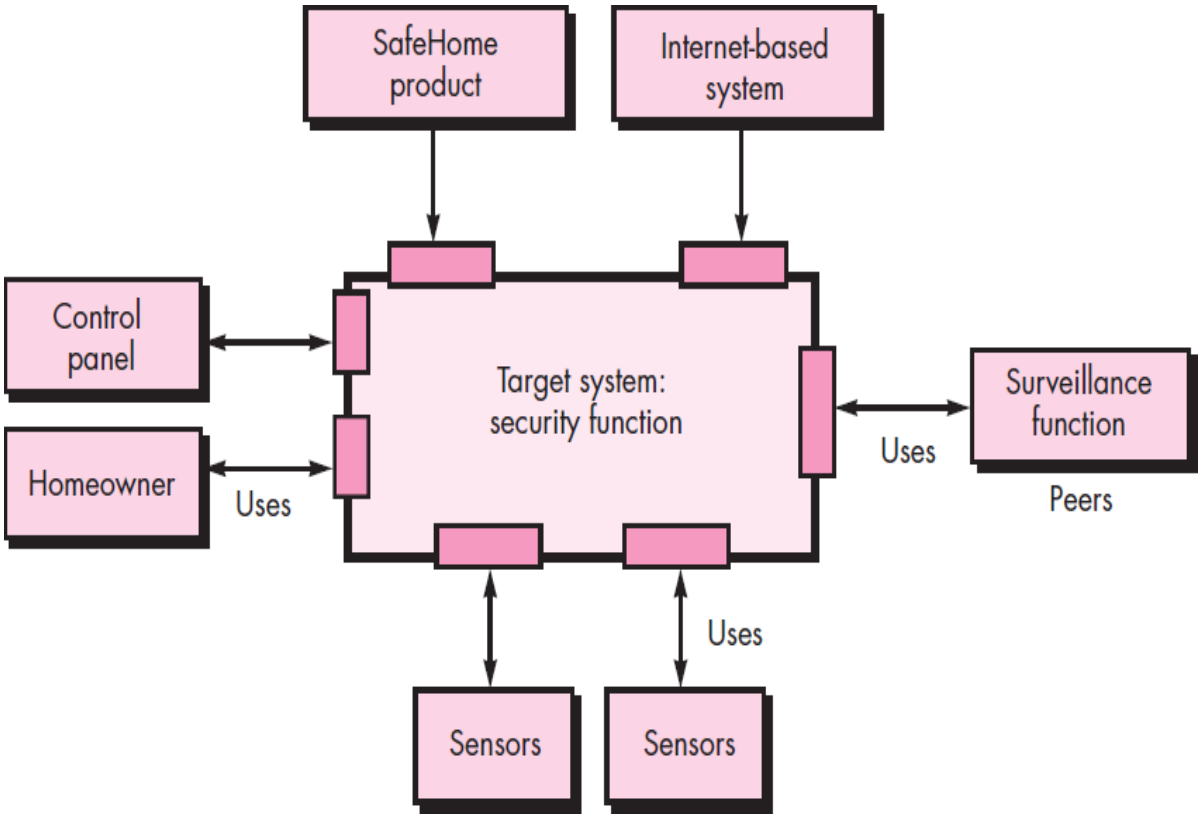
Functional model: Represents the functional hierarchy of a system

Framework model: Attempts to identify repeatable architectural design patterns encountered in similar types of application. This leads to an increase in the level of abstraction.

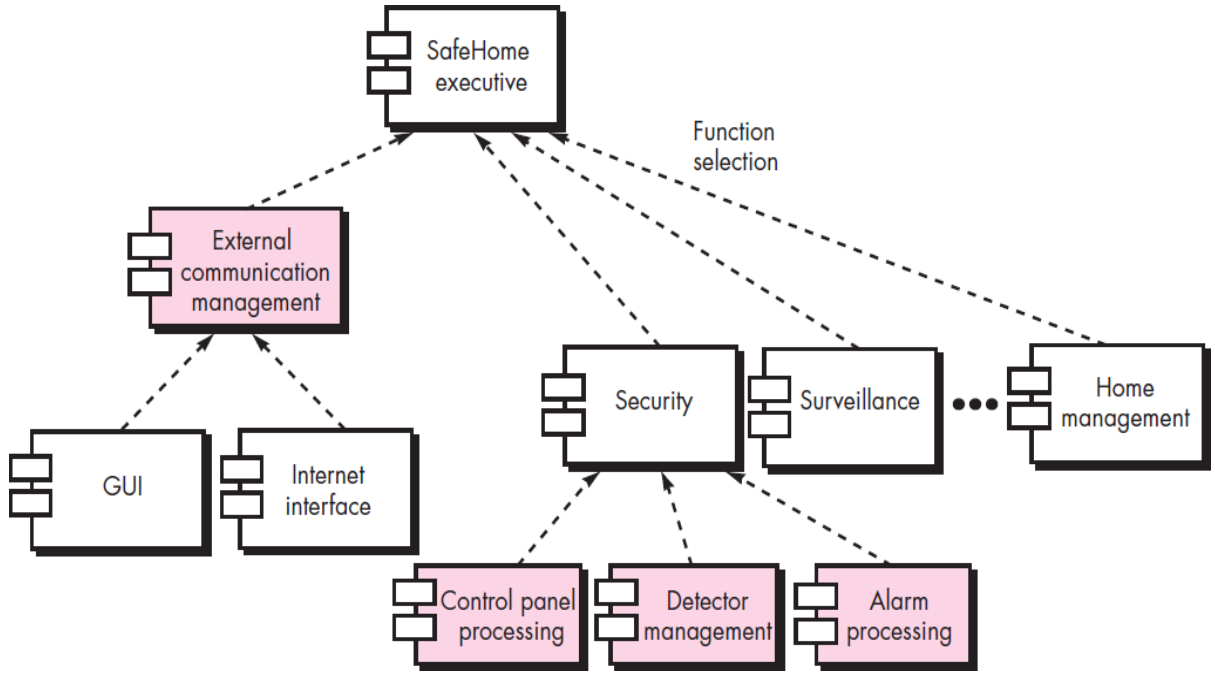
***Architectural Context Diagram*(ACD):**



Architectural Context Diagram for the *SafeHome*



Overall architectural structure for *SafeHome with top-level components*



7.Explain about Architectural Mapping Using Data Flow ?

Architectural Mapping:

A mapping technique, called structured design, is often characterized as a data flow-oriented design method because it provides a convenient transition from a data flow diagram to software architecture.

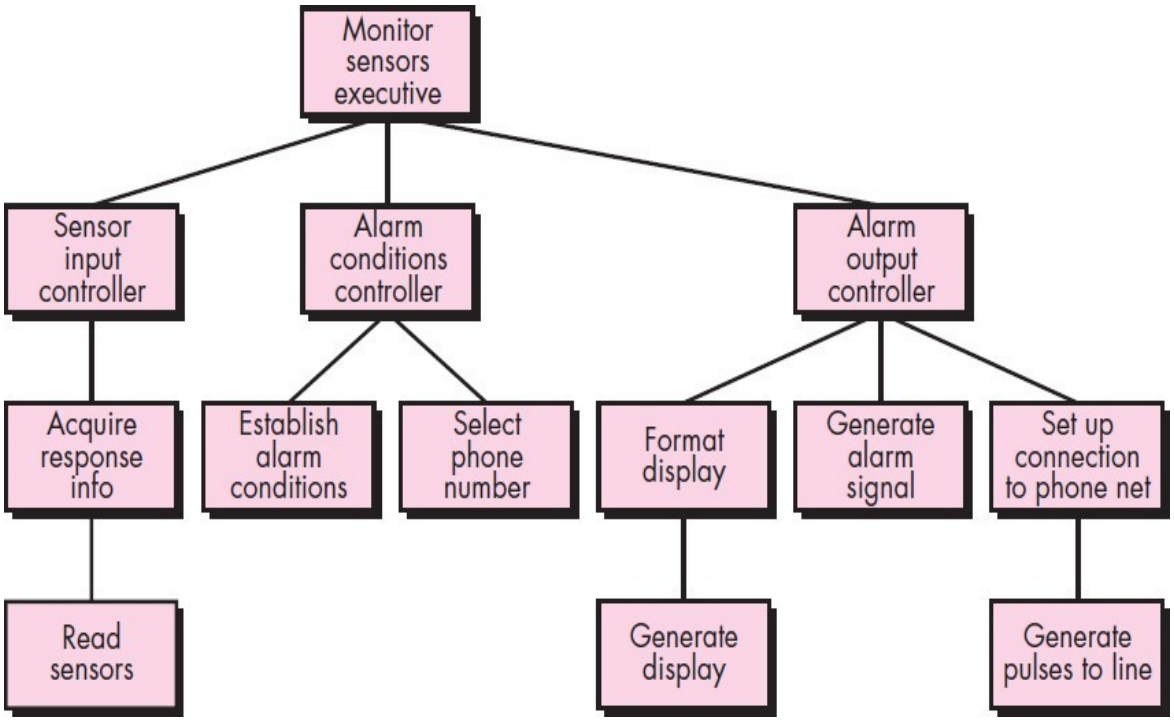
The transition from information flow to program structure is accomplished as part of a six step process:

(1) The type of information flow is established,
(2) Flow boundaries are indicated,
(3) The DFD is mapped into the program structure,
(4) Control hierarchy is defined,
(5) The resultant structure is refined using design measures.
(6) The architectural description is refined and elaborated.

To map these DFD into a software architecture, you would initiate the following design steps:

Step 1. Review the fundamental system model.
Step 2. Review and refine data flow diagrams for the software.
Step 3. Determine whether the DFD has transform or transaction flow characteristics.
Step 4. Isolate the transform center by specifying incoming and outgoing flow boundaries.
Step 5. Perform “first-level factoring.”
Step 6. Perform “second-level factoring.”

Step 7. Refine the first-iteration architecture using design heuristic for improved software quality.



8.What is Component design? Explain about Component-level Design in detailed

Component-Level Design

Component level design is the definition and design of components and modules after the architectural design phase. Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each component for the system development.

A complete set of software components is defined during architectural design. But the internal data structures and processing details of each component are not represented at a level of abstraction that is close to code. Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each component.

According to OMG UML specification component is expressed as,
"A modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces."

Component Views:

• OO View – A component is a set of collaborating classes.
• Conventional View – A component is a functional element of a program that incorporates processing logic, the internal data structures required to implement the processing logic, and an interface that enables the component to be invoked and data to be passed to it.

"Component level design is the definition and design of components and modules after the architectural design phase"

Component-level design defines
The data structures
Algorithms
Interface characteristics
Communication mechanisms allocated to each component for the system development.

Views of an Component

1. An Object-Oriented View
2. The Traditional View
3. A Process-Related View

9.Explain about how designing Class-Based Components ?

Class-Based Component Design:

An individual software component is a software package, a web service, a web resource, or a module that encapsulates a set of related functions (or data).
All system processes are placed into separate components so that all of the data and functions inside each component are semantically related (just as with the contents of classes). Because of this principle, it is often said that components are *modular* and *cohesive*.
With regard to system-wide co-ordination, components communicate with each other via *interfaces*. When a component offers services to the rest of the system, it adopts a *provided* interface that specifies the services that other components can utilize, and how they can do so.
This interface can be seen as a signature of the component - the client does not need to know about the inner workings of the component (implementation) in order to make use of it. This principle results in components referred to as encapsulated. The UML illustrations within this article represent provided interfaces by a lollipop-symbol attached to the outer edge of the component.

Basic Design Principles:

1.The Open-Closed Principle (OCP). *"A module should be open for extension but closed for modification"*

2.The Liskov Substitution Principle (LSP). *"Subclasses should be substitutable for their base classes"*

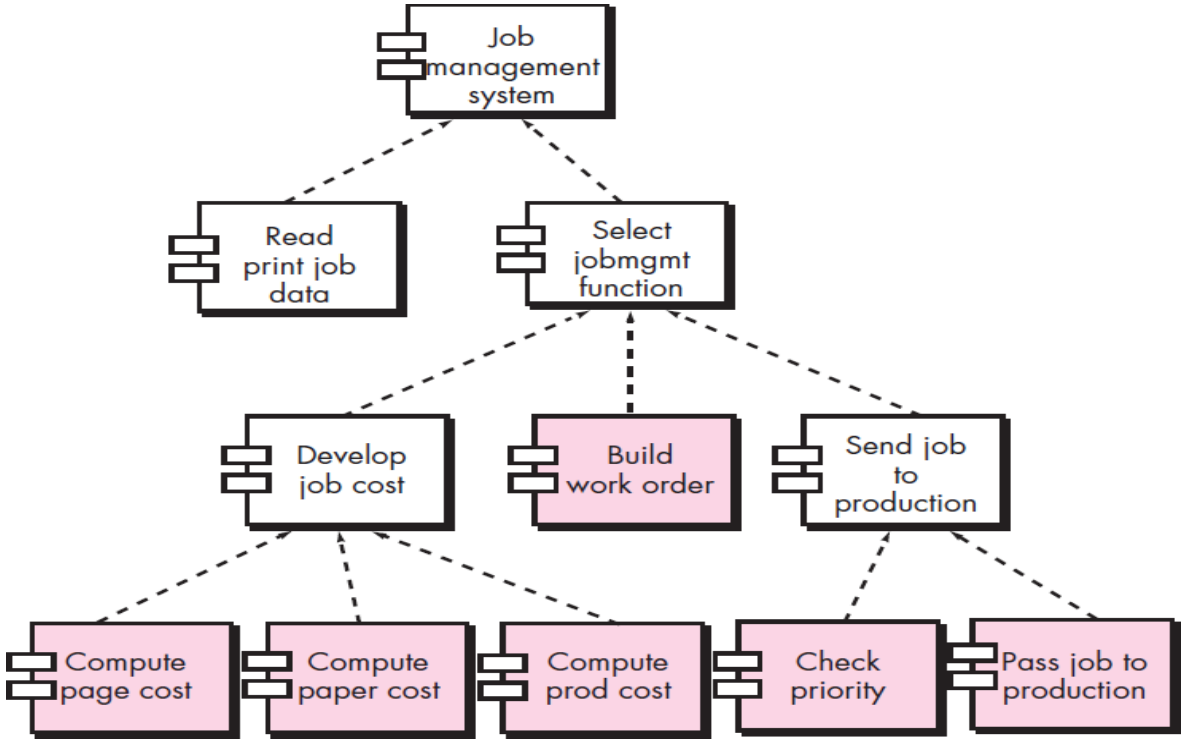
3.Dependency Inversion Principle (DIP). *"Depend on abstractions. Do not depend on concretions"*

*4.*The Interface Segregation Principle (ISP). *"Many client-specific interfaces are better than one general purpose interface"*

*5.*The Release Reuse Equivalency Principle (REP). *"The granule of reuse is the granule of release"*

6.The Common Closure Principle (CCP). *"Classes that change together belong together."*

*7.*The Common Reuse Principle (CRP). *"Classes that aren't reused together should not be grouped together"*



Cohesion:

The "single-mindedness" of a module can be given as cohesion. Cohesion implies that a single component or class encapsulates only attributes and

operations that are closely related to one another and to the class or component itself.

◦ Functional
◦ Layer
◦ Communicational

Functional Cohesion Typically applies to operations where a module performs one and only one computation and then returns a result.

Layer Cohesion applies to packages, components, and classes. It occurs when a higher layer can access a lower layer, but lower layers do not access higher layers. Control Panel provides a good example for layer cohesion, where the higher layer access the lower but the lower layers cant access higher layers.

Communicational Cohesion represents All operations that access the same data are defined within one class. In general, such classes focus solely on the data in question, accessing and storing it.
Example: A Student Record class that adds, removes, updates, and accesses various fields of a student record for client components.

Coupling:

Coupling which provides a qualitative measure of the degree to which classes or components are connected to each other.

Types of Coupling:

Content coupling
Common coupling
Control coupling
Stamp coupling
Data coupling
Routine call coupling
Type use coupling
Inclusion / import coupling
External couplin

10.Discuss about Component Level Design for WebApps ?

Component Level Design for WebApps:

(1) A well-defined cohesive (Interrelated) function that manipulates content or provides computational or data processing for an end user
(2) A cohesive package of content and functionality that provides the

end user with some required capability.

Therefore, component-level design for WebApps often incorporates elements of

1. Content design
2. Functional design.

Content design at the component level focuses on content objects and the manner in which they may be packaged for presentation to a WebApp end user.

The formality of content design at the component level should be tuned to the characteristics of the WebApp to be built.

However, the size and complexity grows, it may be necessary to organize content in a way that allows easier reference and design.

Among many capabilities, the user can select and control

Functional Design at the Component Level

Modern Web applications deliver increasingly sophisticated processing functions that

(1) Perform localized processing to generate content and navigation capability in a dynamic fashion,

(2) Provide computation or data processing capability that is appropriate for the WebApp's business domain,

(3) Provide sophisticated database query and access,

(4) Establish data interfaces with external corporate systems.

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

Step 2. Identify all design classes that correspond to the infrastructure domain.

Step 3. Elaborate all design classes that are not acquired as reusable components.

Step 3a. Specify message details when classes or components collaborate.

Step 3b. Identify appropriate interfaces for each component.

Step 3c. Elaborate attributes and define data types and data structures required to implement them.

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

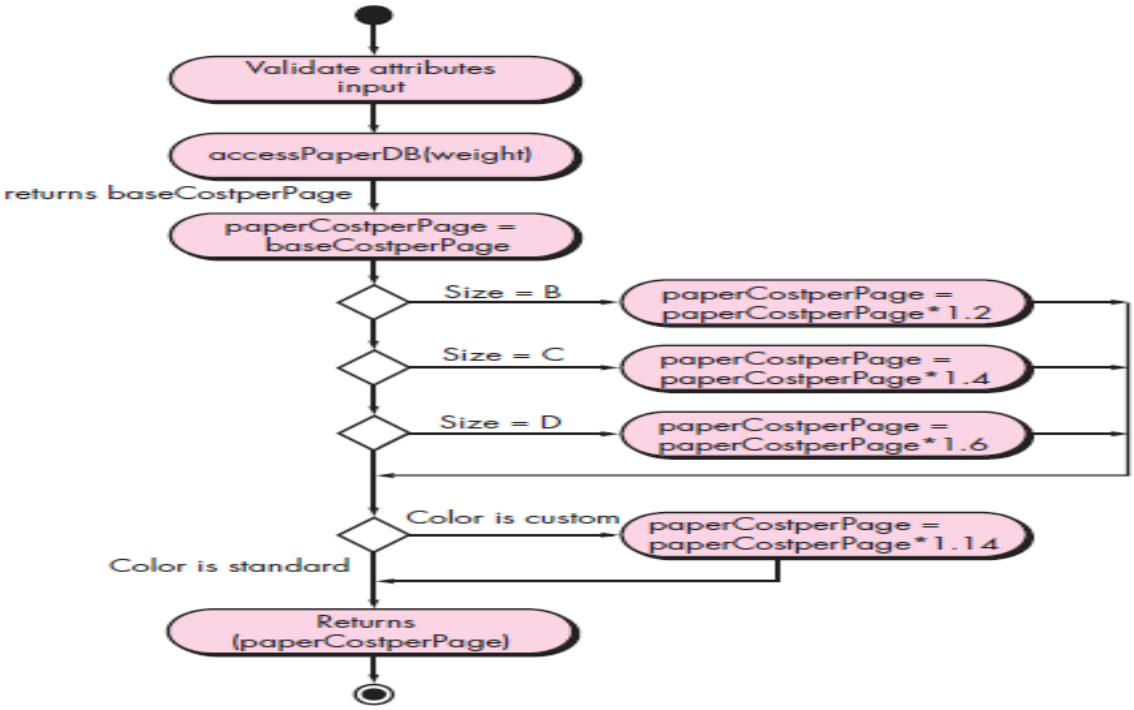
Step 4. Describe persistent data sources (databases and files) and identify the classes required to manage them.

Step 5. Develop and elaborate behavioral representations for a class or component.

Step 6. Elaborate deployment diagrams to provide additional implementation detail.

Step 7. Refactor every component-level design representation and always consider alternatives.

UML activity diagram for *compute- PaperCost()*



11.Explain about how Traditional Components are designed ?

Designing Traditional Components:

Traditional components are designed based on different constructs with are

1. Sequence,
2. Condition,
3. Repetition.

Sequence implements processing steps that are essential in the specification of any algorithm.

Condition provides the facility for selected processing based on some logical occurrence.

Repetition allows for looping.

The use of the structured constructs reduces program complexity and thereby enhances readability, testability, and maintainability.

The use of a limited number of logical constructs also contributes to a human understanding process that psychologists call **chunking**.

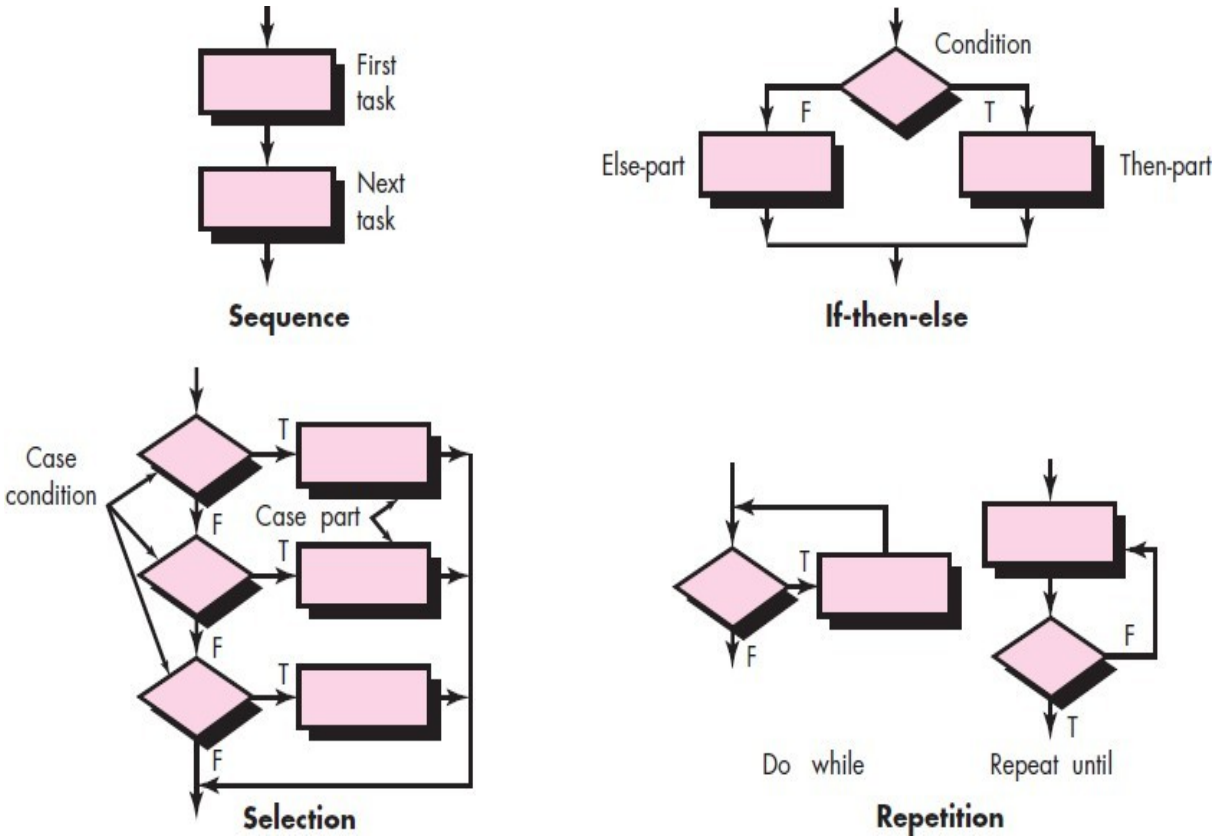
A. Graphical Design Notation

B. Tabular Design Notation

C. Program Design Language

A. Graphical Design Notation

* A picture is worth a thousand words," but it's rather important to know which picture and which 1000 words.
* There is no question that graphical tools, such as the UML activity diagram or the flowchart, provide useful pictorial patterns that readily depict procedural detail.
* The activity diagram allows you to represent sequence, condition, and repetition— all elements of structured programming.
* It is a descendent of an earlier pictorial design representation called a flowchart.
* A flowchart, like an activity diagram, is quite simple pictorially. A box is used to indicate a processing step. A diamond represents a logical condition, and arrows.



B. Tabular Design Notation

* In many software applications, a module may be required to evaluate a complex combination of conditions and select appropriate actions based on these conditions.
* Decision tables provide a notation that translates actions and conditions into a tabular form.

* The table is difficult to misinterpret and may even be used as a machine-readable input to a table-driven algorithm.

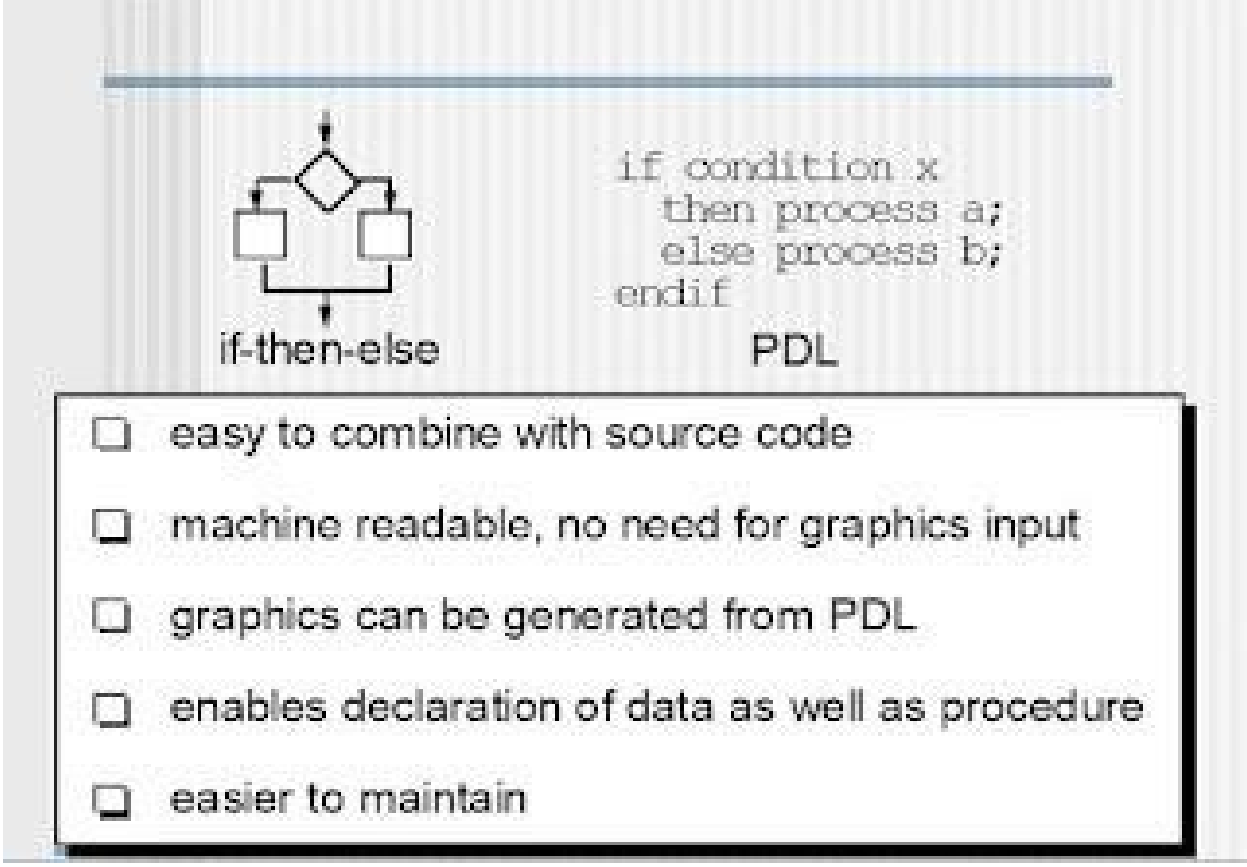
	Rules					
Conditions	**1**	**2**	**3**	**4**	**5**	**6**
Regular customer	T	T				
Silver customer			T	T		
Gold customer					T	T
Special discount	F	T	F	T	F	T
Actions						
No discount	✓					
Apply 8 percent discount			✓	✓		
Apply 15 percent discount					✓	✓
Apply additional *x* percent discount		✓		✓		✓

C. Program Design Language

* Program design language (PDL), also called structured English or pseudo code,
* It incorporates the logical structure of a programming language with the free-form expressive ability of a natural language (e.g., English).
* Narrative text (English) is embedded within a programming language-like syntax.
* Automated tools can be used to enhance the application of PDL.
* A basic PDL syntax should include constructs for

- Component definition,
- Interface description,
- Data declaration,
- Block structuring,
- Condition constructs,

Repetition constructs,
Input-output (I/O) constructs.
It should be noted that PDL can be extended to include keywords.



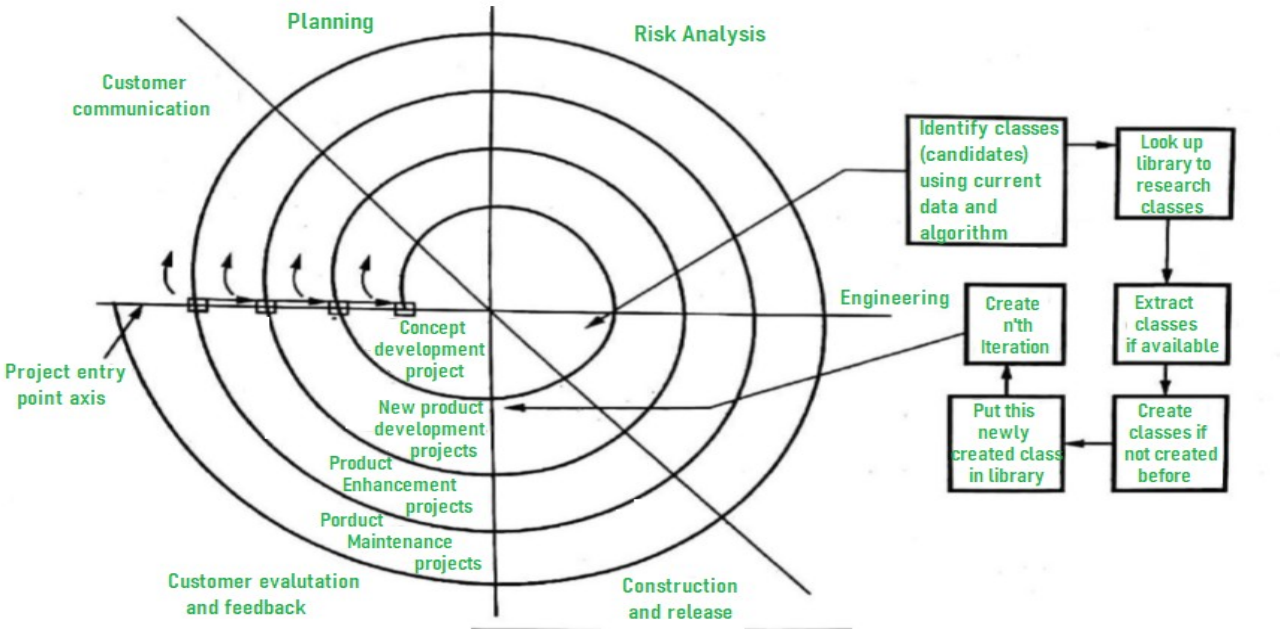
12.Discuss about Component-Based Development ?

Component Based Model (CBM) :

The component-based assembly model uses object-oriented technologies. In object-oriented technologies, the emphasis is on the creation of classes. Classes are the entities that encapsulate data and algorithms. In component-based architecture, classes (i.e., components required to build application) can be uses as reusable components. This model uses various characteristics of spiral model. This model is evolutionary by nature. Hence, software development can be done using iterative approach. In CBD model, multiple classes can be used. These classes are basically the prepackaged components. The model works in following manner:

- **Step-1:** First identify all the required candidate components, i.e., classes with the help of application data and algorithms.

- **Step-2:** If these candidate components are used in previous software projects then they must be present in the library.
- **Step-3:** Such preexisting components can be excited from the library and used for further development.
- **Step-4:** But if the required component is not present in the library then build or create the component as per requirement.
- **Step-5:** Place this newly created component in the library. This makes one iteration of the system.
- **Step-6:** Repeat steps 1 to 5 for creating n iterations, where n denotes the number of iterations required to develop the complete application.



Characteristics of Component Assembly Model:

- Uses object-oriented technology.
- Components and classes encapsulate both data and algorithms.
- Components are developed to be reusable.
- Paradigm similar to spiral model, but engineering activity involves components.
- The system produced by assembling the correct components.

Model Question Bank:

S.NO	QUESTION	CO	BL	MARKS
1	What is Software Design? Explain the Design Concepts in detailed	3	1,1	12
2	How the Software is designed by using the Software Engineering Process	3	2	12
3	Explain about the design process of a software	3	2	12
4	How to develop the Software design model	3	1	12
5	Define Software Architecture? Discuss about Architectural Design in detailed	3	1,2	12
6	Discuss about various Architecture Styles in detailed	3	2	12
7	Explain about Architectural Mapping Using Data Flow	3	1,2	12
8	What is Component design? Explain about Component-level Design in detailed	3	1,2	12
9	Explain about how designing Class-Based Components	3	1	12
10	Discuss about Component Level Design for WebApps	3	2	12
11	Explain about how Traditional Components are designed	3	1	12
12	Discuss about Component-Based Development	3	2	12

Assignment Questions:

S.NO	QUESTION	CO	BL	MARKS
1	What is Software Design? Explain the Design Concepts in detailed	3	1,1	12
2	How to develop the Software design model	3	1	12
3	Define Software Architecture? Discuss about Architectural Design in detailed	3	1,2	12
4	Discuss about various Architecture Styles in detailed	3	2	12
5	Explain about Architectural Mapping Using Data Flow	3	1,2	12
6	What is Component design? Explain about Component-level Design in detailed	3	1,2	12
7	Explain about how designing Class-Based Components	3	1	12
8	Discuss about Component Level Design for WebApps	3	2	12