5. ARCHITECTURAL DESIGN

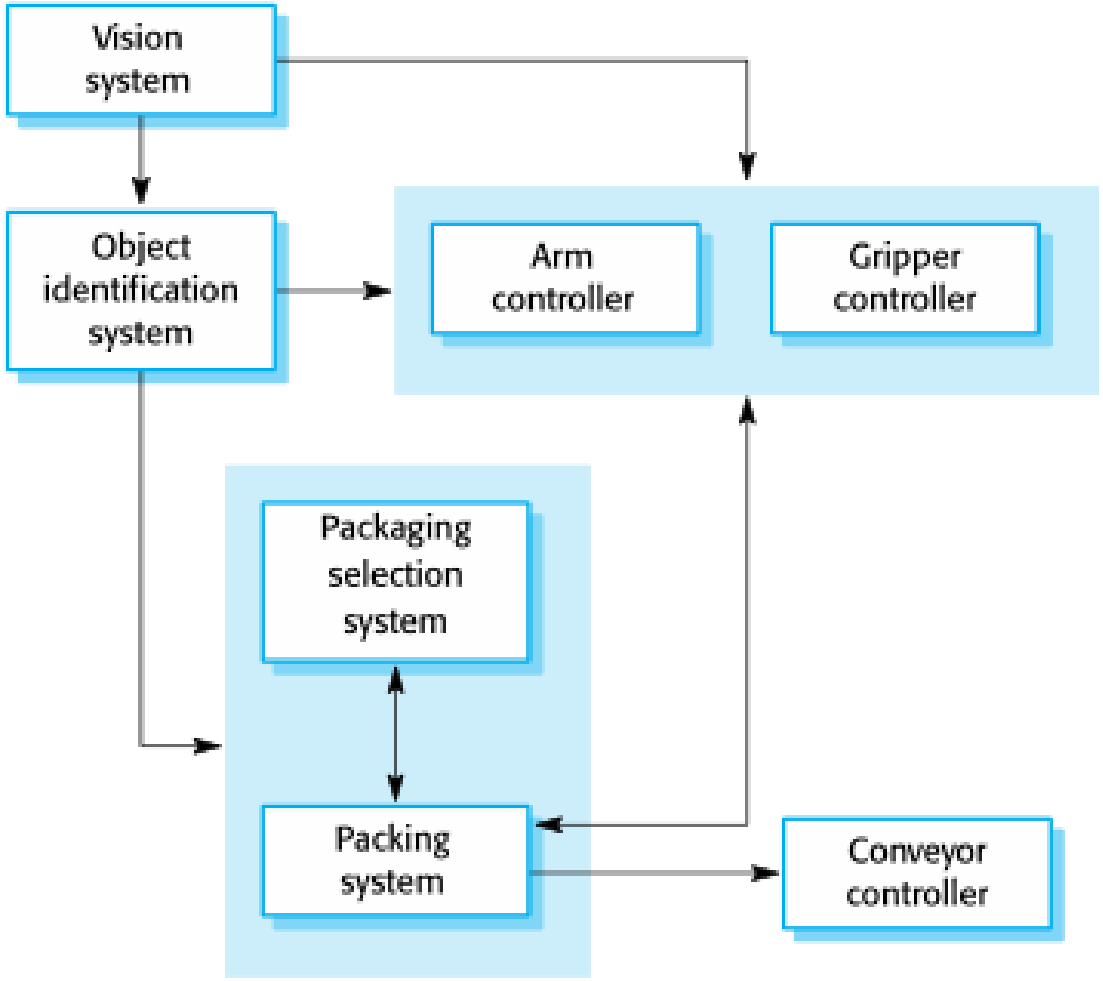
Contents:

5.1 Architectural design decisions

Architectural design

- **Architectural design** is a process for identifying the sub-systems making up a system and the framework for sub-system control and communication.
- Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system.
- The output of this design process is a description of the software architecture that describes how the system is organized as a set of communicating components.
- Architectural design is an early stage of the system design process.
- It represents the link between specification and design processes and is often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications.
- Software architecture is most often represented using simple, informal **block diagrams** showing entities and relationships.

The architecture of a packing robot control system



Abstraction level of Architecture Design

- Software architectures can be designed at **two levels of abstraction**:
- **Architecture in the small** is concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components.
- **Architecture in the large** is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

Advantages of Explicit Architecture design

- Three **advantages** of explicitly designing and documenting software architecture:
  - **Stakeholder communication**: Architecture may be used as a focus of discussion by system stakeholders.
  - **System analysis**: Well-documented architecture enables the analysis of whether the system can meet its non-functional requirements.
  - **Large-scale reuse**: The architecture may be reusable across a range of systems or entire lines of products.

Uses of architectural models:

1. **As a way of facilitating discussion about the system design**
   - A high-level architectural view of a system is useful for communication with system stakeholders and project planning because it is not cluttered with detail.
   - Stakeholders can relate to it and understand an abstract view of the system.
   - They can then discuss the system as a whole without being confused by detail.

2. **As a way of documenting an architecture that has been designed**
   - The aim here is to produce a complete system model that shows the different components in a system, their interfaces and their connections.

Architectural design Decision

Architectural design decisions

- Architectural design is a **creative process.** So, the process differs depending on the type of system being developed.
- However, a number of **common decisions** span all design processes and these decisions affect the non-functional characteristics of the system:
  - Is there a generic application architecture that can be used?
  - How will the system be distributed?
  - What architectural styles are appropriate?
  - What approach will be used to structure the system?
  - How will the system be decomposed into modules?
  - What control strategy should be used?
  - How will the architectural design be evaluated?
  - How should the architecture be documented?

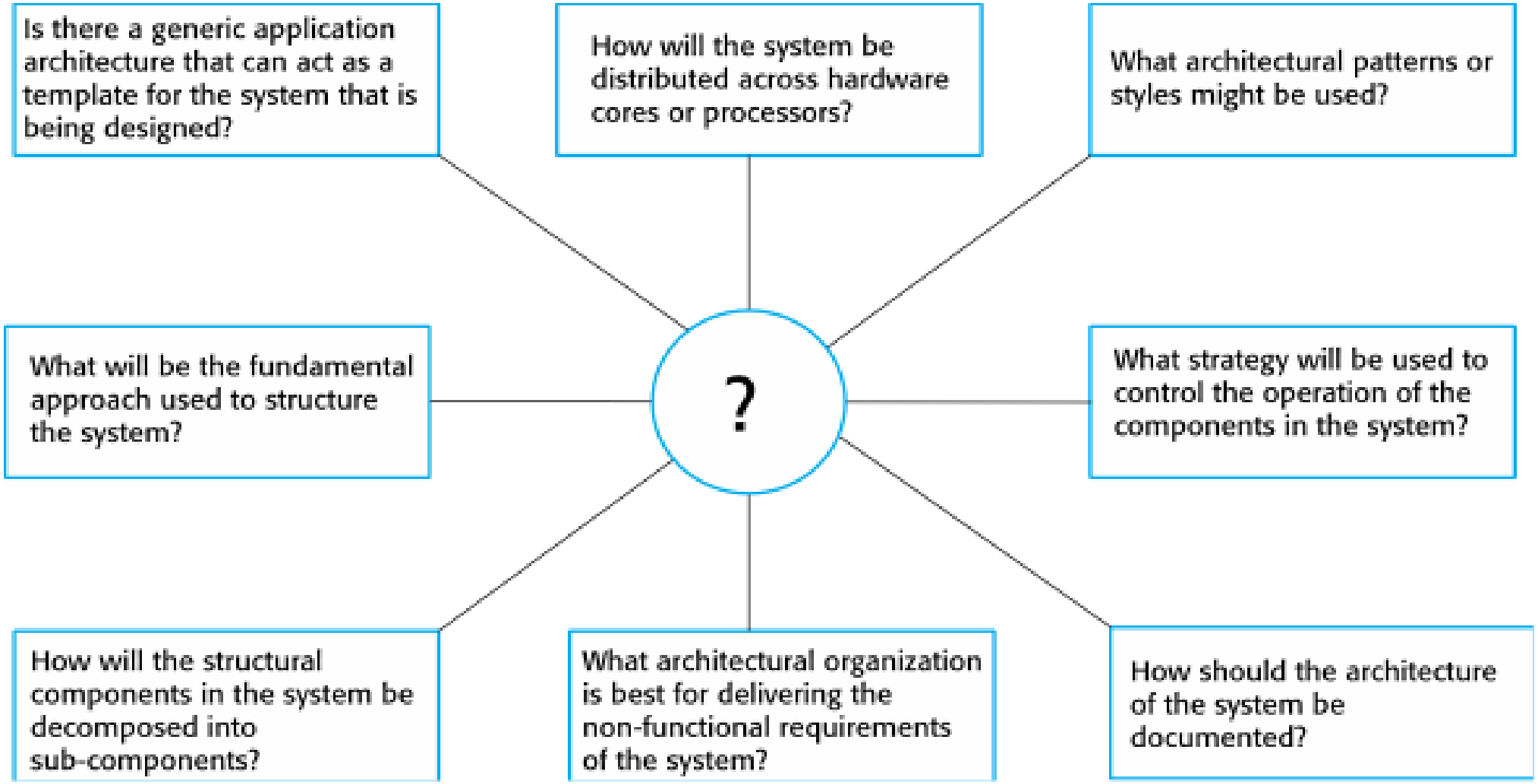


Figure: Architectural design decisions

Factors to be considered while designing architecture:

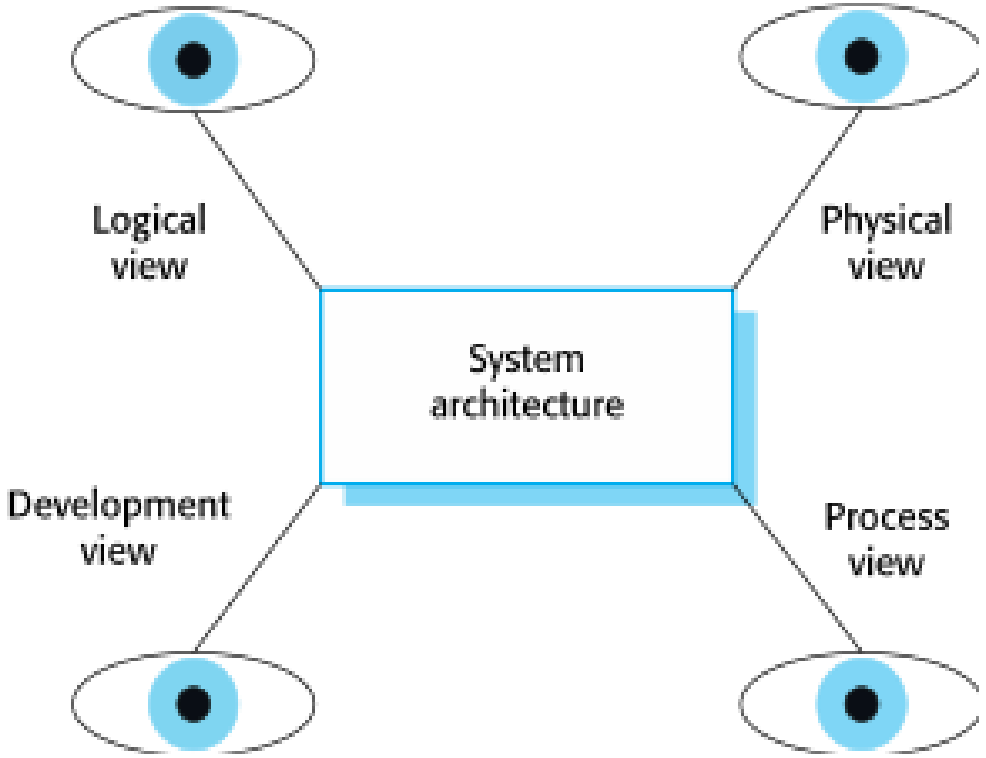
- Because of the close relationship between non-functional system characteristics and software architecture, the choice of architectural style and structure should depend on the non-functional requirements of the system.
  1. **Performance**: If performance is a critical requirement, localize critical operations and minimize communications. Use large rather than fine-grain components.
  2. **Security**: If security is a critical requirement, a layered structure for the architecture should be used. The most critical assets should be protected in the innermost layers and a high level of security validation applied to these layers.
  3. **Safety**: If safety is a critical requirement, the architecture should be designed so that safety-related operations are co-located in a single component or in a small number of components.
  4. **Availability**: If availability is a critical requirement, the architecture should be designed to include redundant components so that it is possible to replace and update components without stopping the system
  5. **Maintainability**: If maintainability is a critical requirement, the system architecture should be designed using fine-grain, self-contained components that may readily be changed.

Architectural views

- Each architectural model only shows **one view or perspective** of the system.
- It might show how a system is decomposed into modules, how the run-time processes interact or the different ways in which system components are distributed across a network.
- For both design and documentation, you usually need to present multiple views of the software architecture.
- In software engineering, architectural views provide different perspectives on a system, allowing stakeholders to understand and analyze specific aspects of the architecture.
- These views help in addressing concerns of different stakeholders, such as developers, system architects, project managers, and end-users.

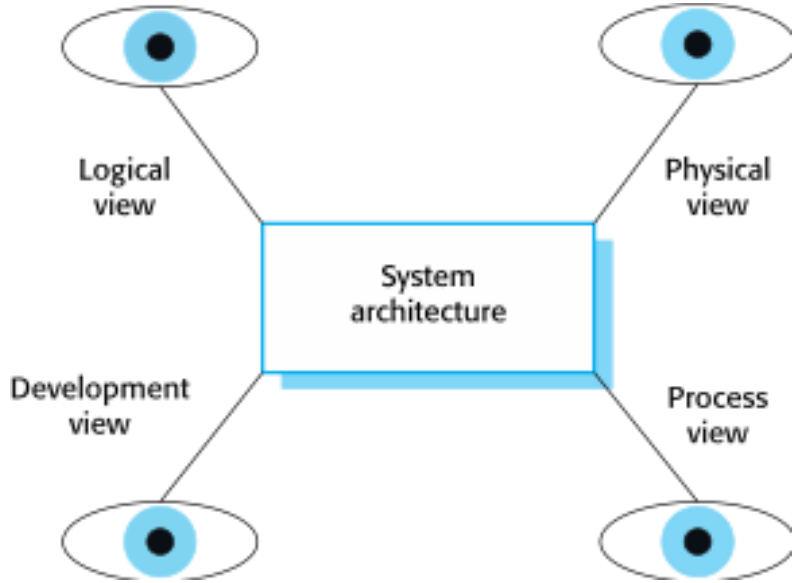
Architectural views

- The 4+1 Architectural View Model is a framework for organizing different perspectives of a software system architecture.
- It was introduced by Philippe Kruchten and is commonly referred to as the "4+1" model because it consists of four primary views, plus a set of scenarios that describe the system's behavior.
- The goal is to provide a comprehensive understanding of the architecture from various angles.



Architectural views

1. **Logical view:**
   - Focuses on the functionality provided by the system from the user's perspective.
   - Describes the high-level components, their relationships, and how they collaborate to achieve the system's objectives.
   - Often represented using class diagrams, package diagrams, or other modeling notations.
2. **Process View:**
   - Emphasizes the dynamic aspects of the system, describing the processes or threads that execute concurrently.
   - Illustrates how tasks are scheduled, how processes communicate, and how concurrency and synchronization are managed.
   - Typically uses diagrams like activity diagrams, state diagrams, or sequence diagrams.



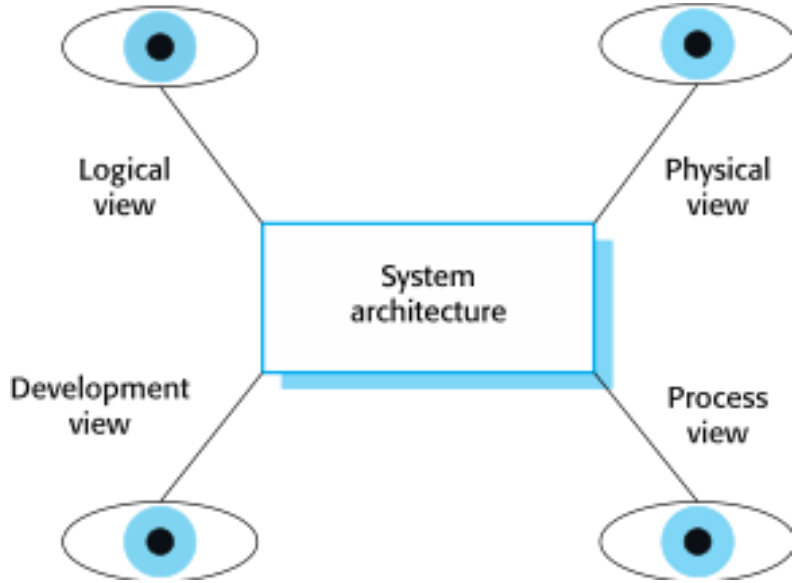
Architectural views

3. Physical View:

- Concentrates on the system's physical architecture, detailing how the software is mapped onto the hardware.
- Describes the distribution of components, nodes, and the physical communication infrastructure.
- May involve deployment diagrams, network diagrams, or other relevant representations.

4. Development View:

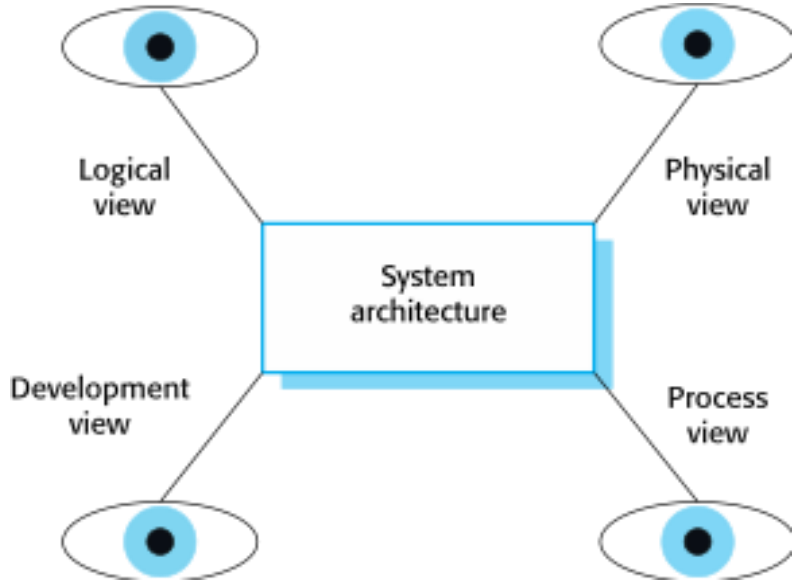
- Provides insight into the organization and structure of the software from the developer's perspective.
- Describes how the software is decomposed into modules, components, or layers and how development tasks are allocated.
- Uses component diagrams, package diagrams, or other representations to show the software's structure.



Architectural views

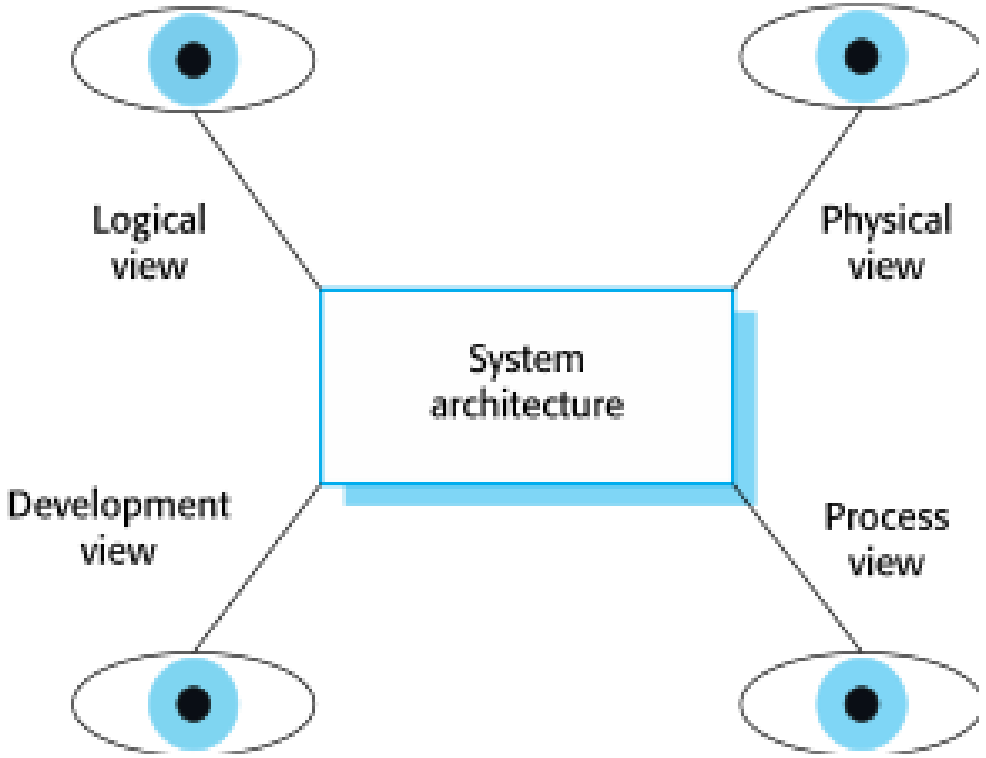
- Additionally, there is the "+1" aspect of the model:

5. **Scenarios (Use Cases):**
   - Describes a set of representative scenarios or use cases that illustrate how the system behaves under various conditions.
   - Helps validate and refine the architecture by providing concrete examples of system behavior.
   - May use of usecase diagrams, sequence diagrams, or other scenario-based representations.



Summary:

- **4+1 view** model of software architecture:
  1. A **logical** view:
     - shows the key abstractions in the system as objects or object classes.
  2. A **process** view
     - shows how, at run-time, the system is composed of interacting processes.
  3. A **development** view
     - shows how the software is decomposed for development.
  4. A **physical** view
     - shows the system hardware and how software components are distributed across the processors in the system.
  5. Related using **use cases** or scenarios (+1).



Architectural Patterns

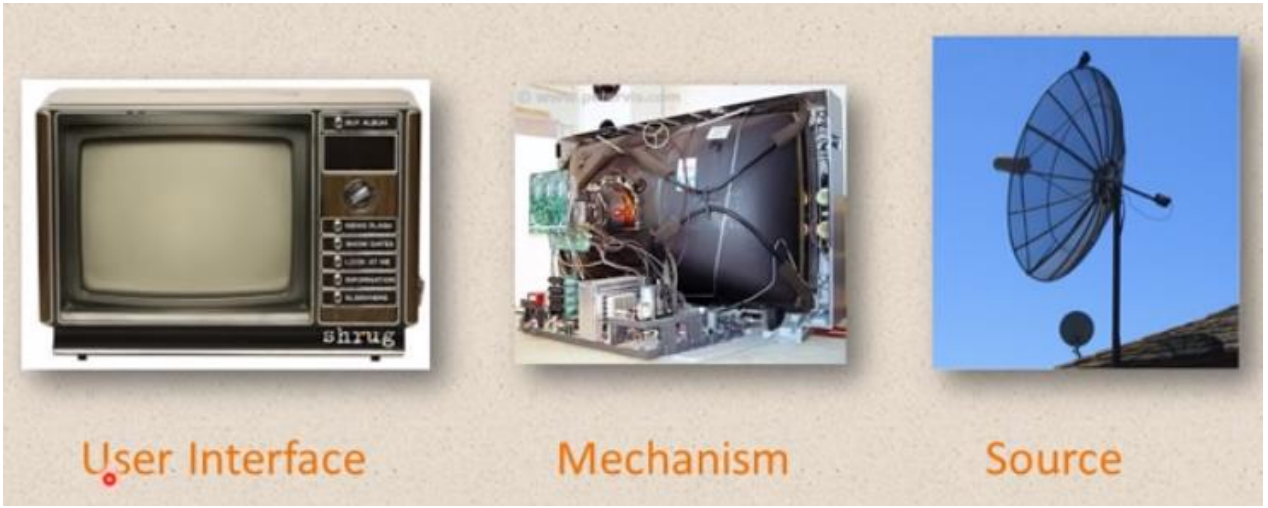
Architectural patterns

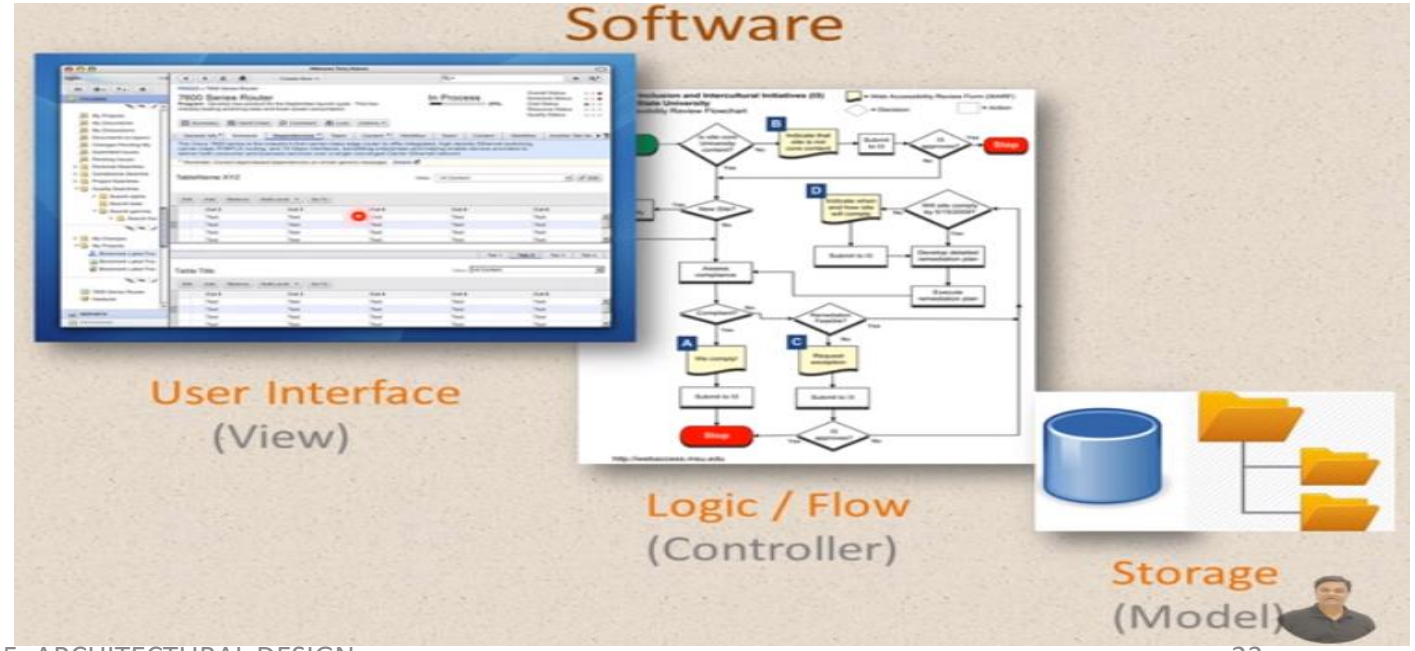
- Patterns are a means of representing, sharing and reusing knowledge.
- An architectural pattern is a stylized description of a good design practice, which has been tried and tested in different environments.
- Patterns should include information about when they are and when the are not useful.
- Patterns may be represented using tabular and graphical descriptions.
- Some popularly used architectural patterns are:
    1. Model-View-Controller (MVC) Architecture
    2. Layered Architecture
    3. Client-Server Architecture
    4. Repository Architecture
    5. Pipe and Filter Architecture

5.2 Model-View-Controller architectural pattern

Any Engineered products have:

- User Interface
- Mechanism
- Input/Source/Storage





MVC architecture

- Separates presentation and interaction from the system data.
- Supported by most language frameworks.
- The system is structured into three logical components that interact with each other.

1. The Model

- It is the central component of the pattern that directly manages the data, logic and rules of the application.
- It is the application's dynamic data structure, independent of the user interface.
- manages the system data and associated operations on that data.

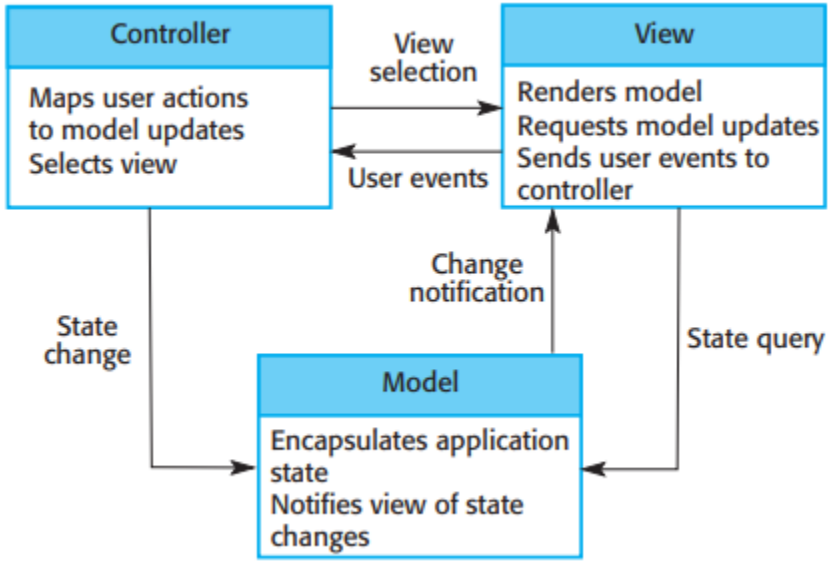


Figure: The organization of the Model-View-Controller

2. The View

- The **View** component defines and manages how the data is presented to the user.
- can be any output representation of information, such as a chart or a diagram.
- Multiple views of the same information are possible.

3. The controller

- accepts input and converts it to commands for the model or view.
- It manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model.

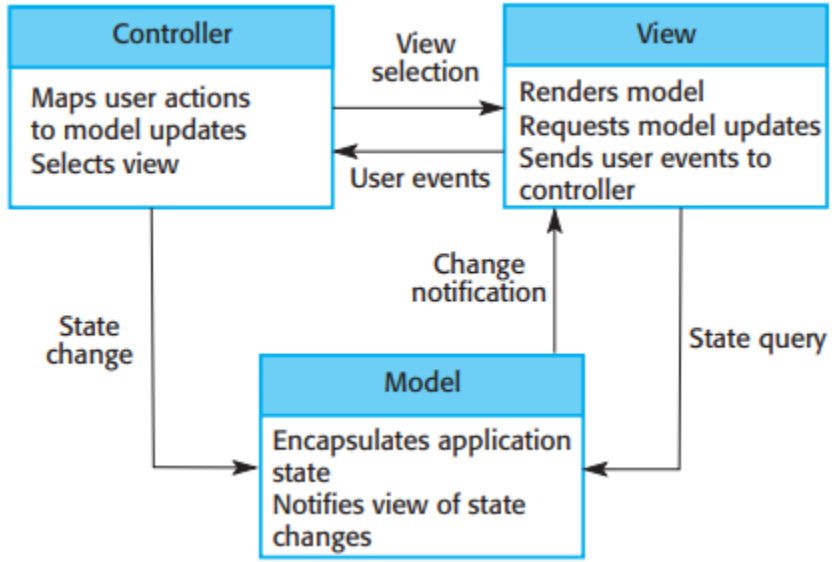
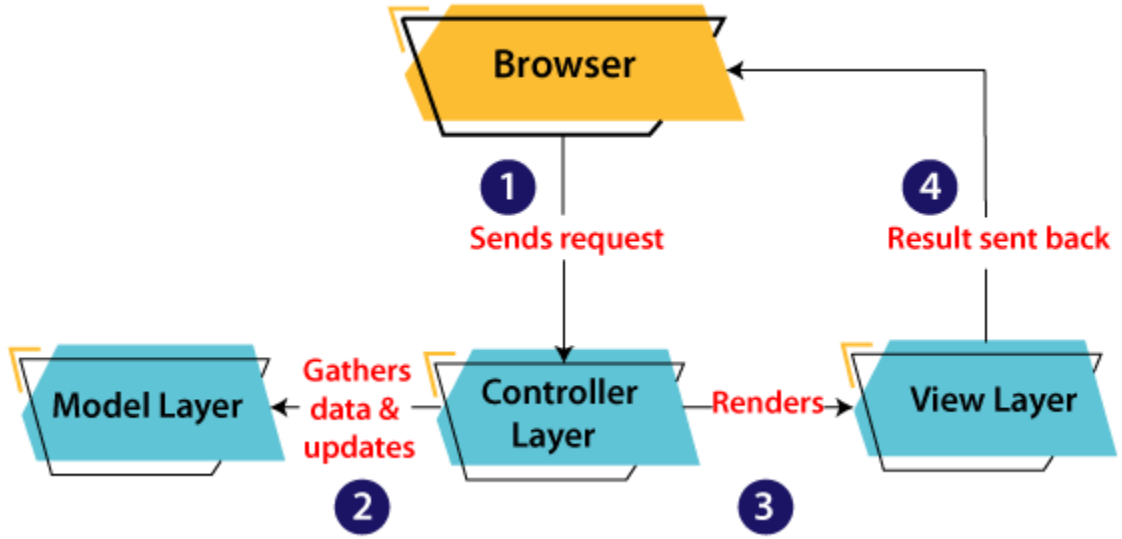


Figure: The organization of the Model-View-Controller

MVC architecture in web

- Serves as a basis of interaction management in many web-based systems.
    1. A client (browser) sends a request to the controller on the server side, for a page.
    2. The controller then calls the model. It gathers the requested data.
    3. Then the controller transfers the data retrieved to the view layer.
    4. Now the result is sent back to the browser (client) by the view.



Name	MVC (Model-View-Controller)
Description	• *Separates presentation and interaction from the system data.* • *The system is structured into three logical components that interact with each other.* • *The Model component manages the system data and associated operations on that data.* • *The View component defines and manages how the data is presented to the user.* • *The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model.*
When used	• *Used when there are multiple ways to view and interact with data.* • *Also used when the future requirements for interaction and presentation of data are unknown.*
Advantages	• *Allows the data to change independently of its representation and vice versa.* • *Supports presentation of the same data in different ways with changes made in one representation shown in all of them.*
Disadvantages	• *Can involve additional code and code complexity when the data model and interactions are simple.*

5.3 Layered architectural pattern

Layered Architecture

- Layered architecture is a software architectural pattern that organizes the structure of a system into multiple layers, each responsible for a specific set of functionality.
- The layered architecture style is one of the most common architectural styles.
- The idea behind Layered Architecture is that modules or components with similar functionalities are organized into horizontal layers. As a result, each layer performs a specific role within the application.
- A layer is a logical separation of components or code.
- In these frameworks, components that are related or that are similar are usually placed on the same layers. However, each layer is different and contributes to a different part of the overall system.

The four layers:

1. **Presentation Layer:**
   - The topmost layer responsible for presenting information to the user and handling user input.
   - Includes user interfaces, screens, and other components that interact directly with the end-user.
2. **Application Layer (User Interface Management):**
   - Security and permission rules must be implemented at this layer as well.
   - Performs tasks such as validation, processing, and business rule enforcement.

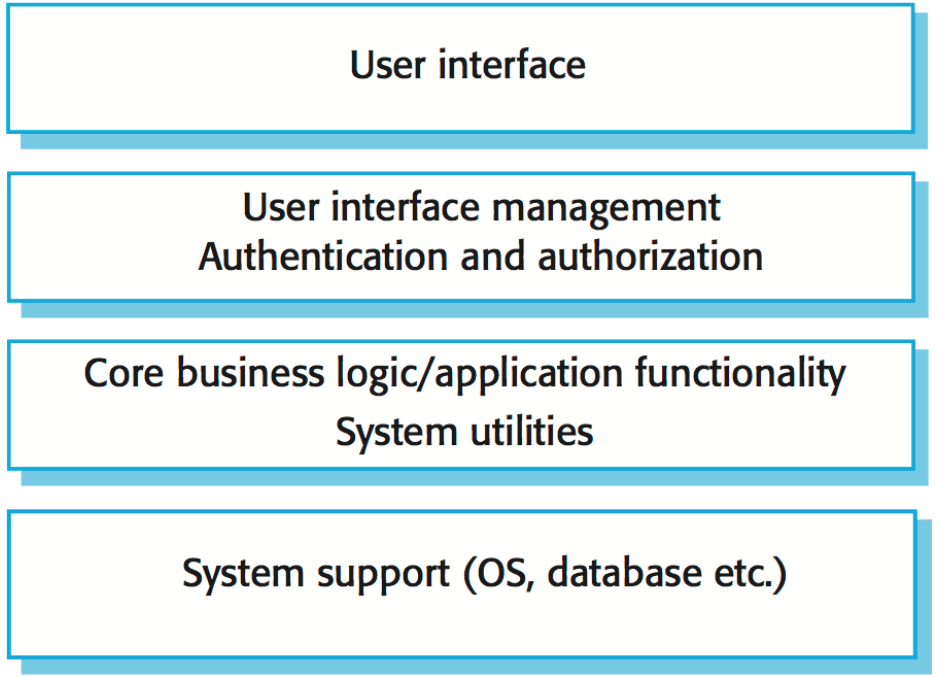


Figure: A generic layered architecture

The four layers:

3. **Domain Layer:**
   - Focuses on representing the core business entities and encapsulating business rules.
   - Typically involves classes and entities related to the business domain.
4. **Infrastructure (or Data Access) Layer:**
   - Manages the interaction with data storage systems, such as databases or external services.
   - Responsible for data retrieval, storage, and communication with external systems.

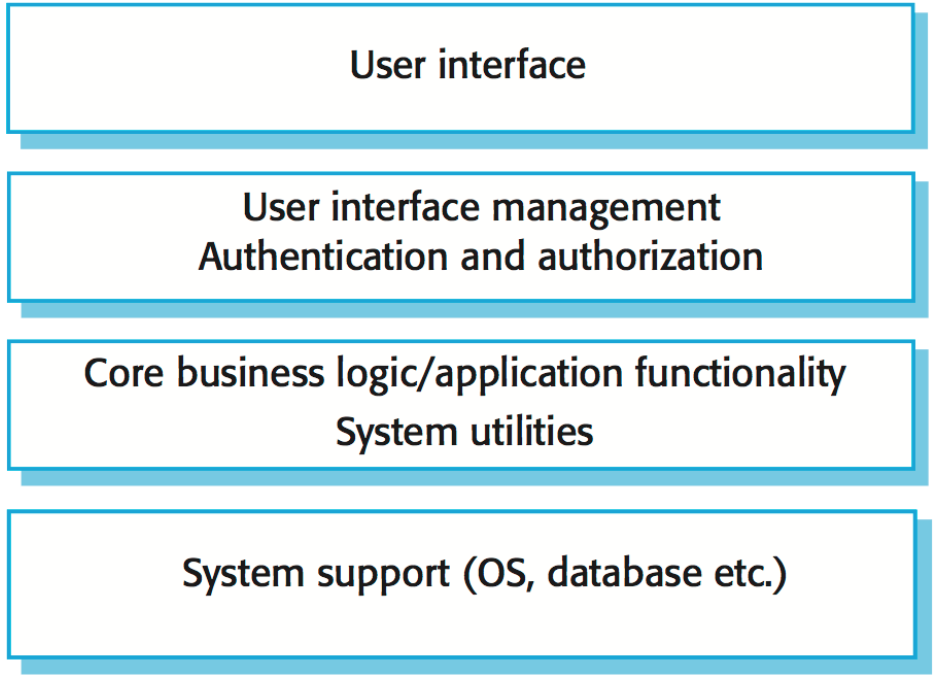


Figure: A generic layered architecture

Example: MentCare System Layered Architecture

1. The top layer is a browser-based user interface.
2. The second layer provides the user interface functionality that is delivered through the web browser.
3. The third layer implements the functionality of the system and provides components that implement system security, patient information creation and updating, import and export of patient data from other databases, and report generators that create management reports.
4. Finally, the lowest layer, which is built using a commercial database management system, provides transaction management and persistent data storage.

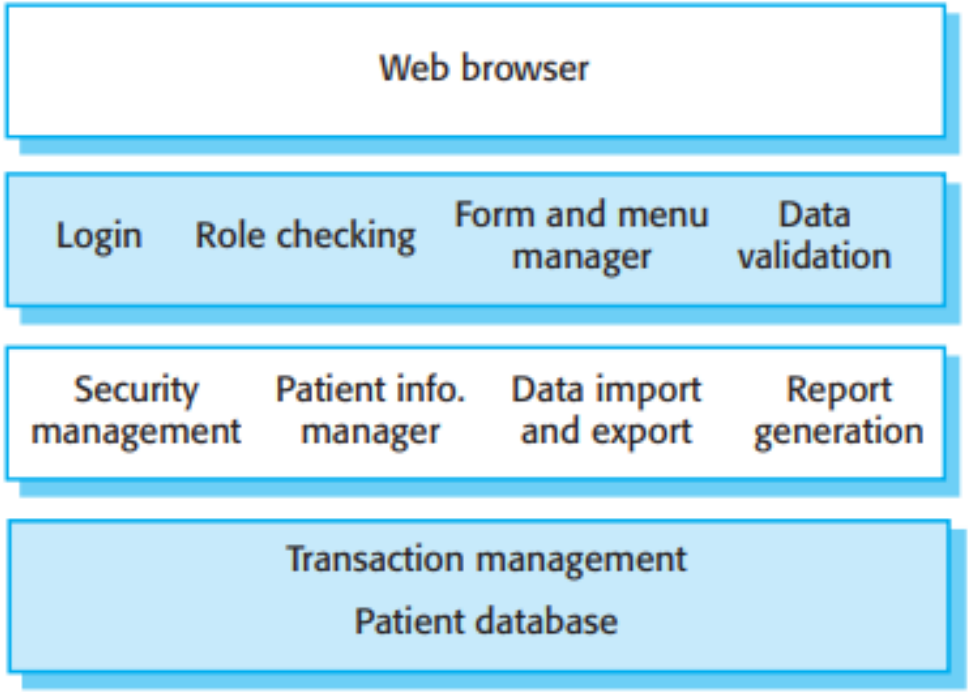


Figure: The architecture of the Mentcare system

Name	Layered architecture
Description	• *Organizes the system into layers with related functionality associated with each layer.* • *A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system.*
When used	• *Used when building new facilities on top of existing systems;* • *when the development is spread across several teams with each team responsibility for a layer of functionality;* • *when there is a requirement for multi-level security.*
Advantages	• *Allows replacement of entire layers so long as the interface is maintained.* • *Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.*
Disadvantages	• *In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it.* • *Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.*

5.5 Client-server architecture

Client-Server Architecture

- Client-server architecture is a widely used software architectural pattern in which the application or system is divided into two main components: the client and the server.
- These components interact with each other over a network, typically using communication protocols such as HTTP, TCP/IP, or others.
- The client is responsible for user interface and user input, while the server handles the business logic, data storage, and processing.

Components of Client-server architecture

1. **A set of servers**
   - that offer services to other components.
   - Servers are software components, and several servers may run on the same computer.
   - Examples of servers include print servers that offer printing services, file servers that offer file management services, and a compile server that offers programming language compilation services.
2. **A set of clients:**
   - that call on the services offered by servers.
   - There will normally be several instances of a client program executing concurrently on different computers.
3. **A network**
   - that allows the clients to access these services.
   - Client–server systems are usually implemented as distributed systems, connected using Internet protocols

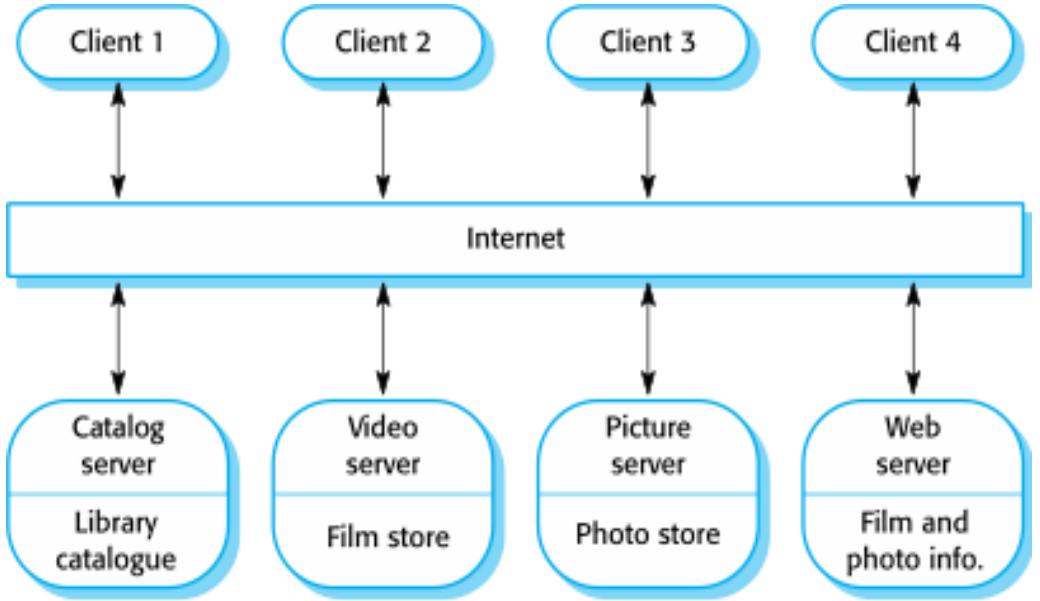


Figure: A client– server architecture for a film library

Name	Client-server
Description	• *In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server.* • *Clients are users of these services and access servers to make use of them.*
When used	• *Used when data in a shared database has to be accessed from a range of locations.* • *Because servers can be replicated, may also be used when the load on a system is variable.*
Advantages	• *The principal advantage of this model is that servers can be distributed across a network.* • *General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.*
Disadvantages	• *Each service is a single point of failure so susceptible to denial of service attacks or server failure.* • *Performance may be unpredictable because it depends on the network as well as the system.* • *May be management problems if servers are owned by different organizations.*

Assignment:

1. List out the advantages and disadvantages of:
   a) MVC architecture
   b) Layered Architecture
   c) Client-Server Architecture

   Note: Search in the internet and prepare it.

Key points:

1. A software architecture is a description of how a software system is organized. Properties of a system such as performance, security, and availability are influenced by the architecture used.
2. Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used, and the ways in which the architecture should be documented and evaluated.
3. Architectures may be documented from several different perspectives or views. Possible views include a conceptual view, a logical view, a process view, a development view, and a physical view.
4. Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used, and point out its advantages and disadvantages.
5. Commonly used Architectural patterns include model-view-controller, layered architecture, repository, client–server, and pipe and filter

References:

1. Architectural Design
https://www.cs.ccsu.edu/~stan/classes/CS410/Notes16/06-ArchitecturalDesign.html#:~:text=Architectural%20design%20is%20a%20process,of%20the%20system%20design%20process.

2. Text Book: Sommerville, Software Engineering, 10 ed., Chapter 6

Brief Answer Questions:

1. Define Architectural design.
2. Mention any two uses of architectural models.
3. What are the factors to be considered while designing architecture of a system? List them.
4. List down the 4+1 architectural views.
5. Define architectural pattern. List any three patterns.
6. What is a MVC pattern? Explain MVC architectural pattern with the design principles it helps to adhere.
7. Write any two advantages and two disadvantages of MVC architecture.
8. What is a layered architecture?
9. What are the layers of layered architectural pattern?
10. What are client and servers? Explain the mechanism in Client-server architecture.

Short Answer Questions:

1. Explain Architectural design and its importance in software engineering.
2. How architectural design decisions are made?
3. Explain 4+1 architectural views.
4. What the roles of each blocks in MVC architecture? Explain with a block diagram.
5. Explain the communication mechanism with MVC architecture in web.
6. What is layered architecture? Explain the four layers.
7. What is client-server architecture? Explain.

Exercise from Book:

1. When describing a system, explain why you may have to start the design of the system architecture before the requirements specification is complete.
2. You have been asked to prepare and deliver a presentation to a nontechnical manager to justify the hiring of a system architect for a new project. Write a list of bullet points setting out the key points in your presentation in which you explain the importance of software architecture.
3. Performance and security may pose to be conflicting non-functional requirements when architecting software systems. Make an argument in support of this statement.
4. Draw diagrams showing a conceptual view and a process view of the architectures of the following systems:
   - A ticket machine used by passengers at a railway station.
   - A computer-controlled video conferencing system that allows video, audio, and computer data to be visible to several participants at the same time.
   - A robot floor-cleaner that is intended to clean relatively clear spaces such as corridors. The cleaner must be able to sense walls and other obstructions.
5. A software system will be built to allow drones to autonomously herd cattle in farms. These drones can be remotely controlled by human operators. Explain how multiple architectural patterns can fit together to help build this kind of system.

Exercise from Book:

6. Suggest an architecture for a system (such as iTunes) that is used to sell and distribute music on the Internet. What Architectural patterns are the basis for your proposed architecture?
7. An information system is to be developed to maintain information about assets owned by a utility company such as buildings, vehicles, and equipment. It is intended that this will be updatable by staff working in the field using mobile devices as new asset information becomes available. The company has several existing asset databases that should be integrated through this system. Design a layered architecture for this asset management system based on the generic information system architecture shown in Figure 6.18
8. Using the generic model of a language processing system presented here, design the architecture of a system that accepts natural language commands and translates these into database queries in a language such as SQL.
9. Using the basic model of an information system, as presented in Figure 6.18, suggest the components that might be part of an information system that allows users to view box office events, available tickets and prices, and to eventually buy tickets.
10. Should there be a separate profession of 'software architect' whose role is to work independently with a customer to design the software system architecture? A separate software company would then implement the system. What might be the difficulties of establishing such a profession?

End of Chapter