

# CS 123

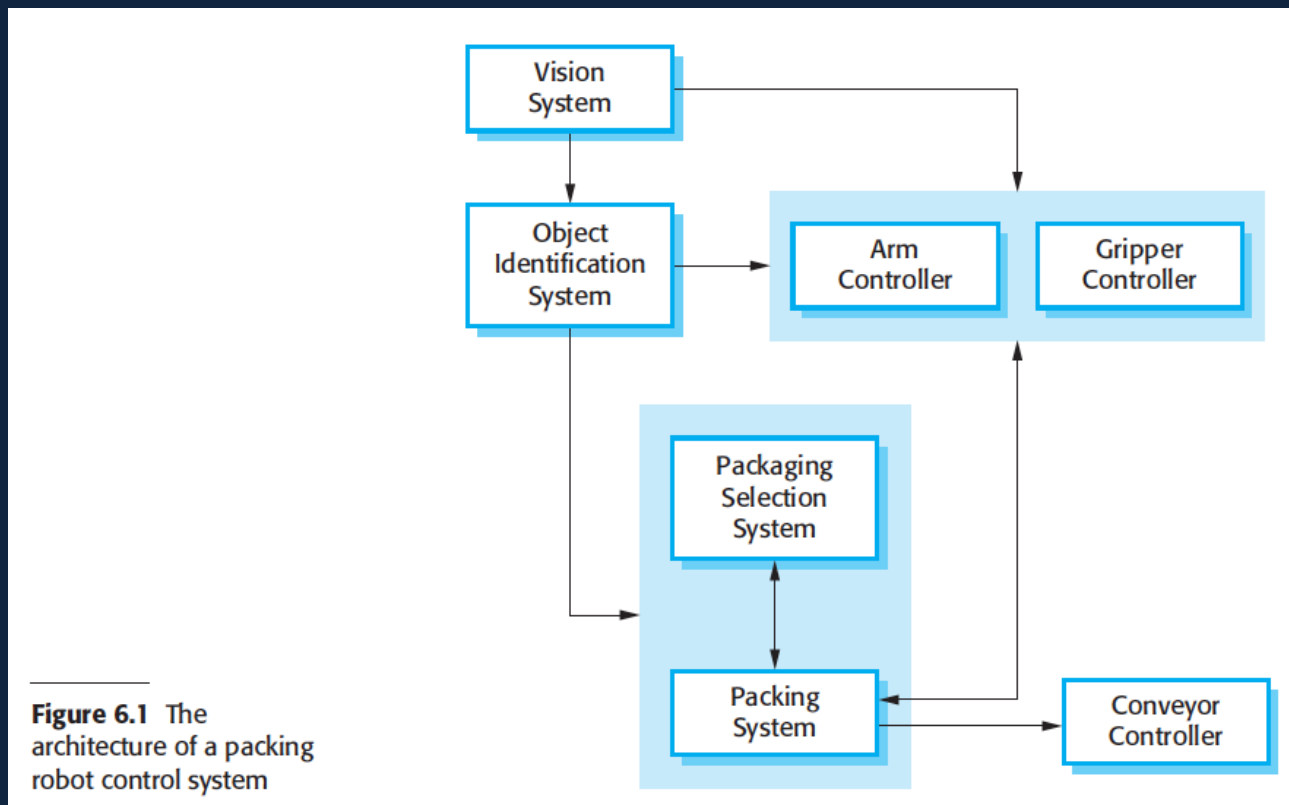
## Introduction to Software Engineering

### 06-B: Architectural Design

DISCS  
SY 2013-2014

# Architectural Design

- Architectural Design is concerned with understanding how a system should be organized and designing an overall structure of that system.



# Software Architecture

- (Bosch, 2000) Software Architecture is important because it affects the system's
  - Performance
  - Robustness
  - Distributability
  - Maintainability

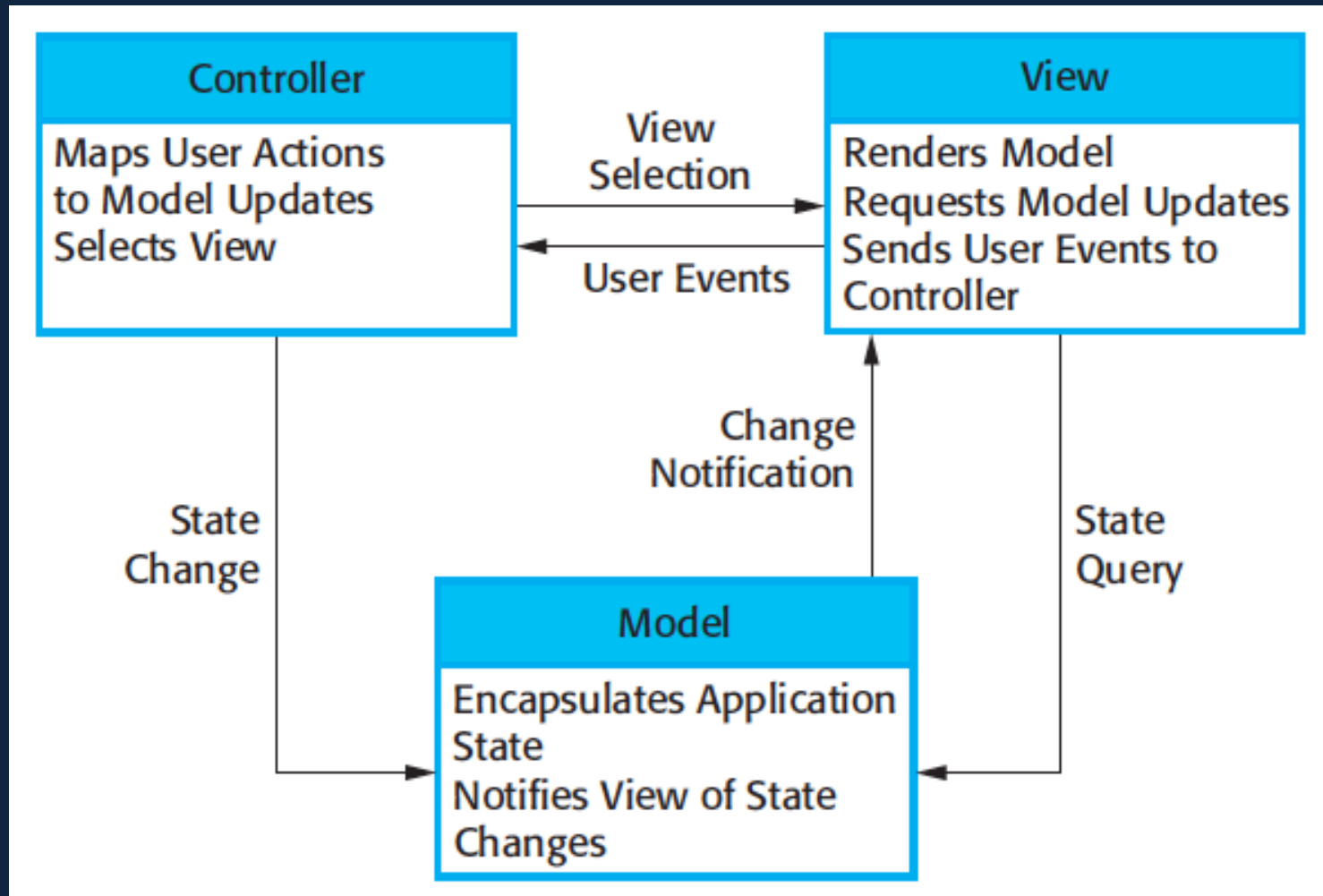
# Software Architecture

- (Bosch, 2000) Software Architecture is important because it affects the system's
  - Performance
  - Robustness
  - Distributability
  - Maintainability

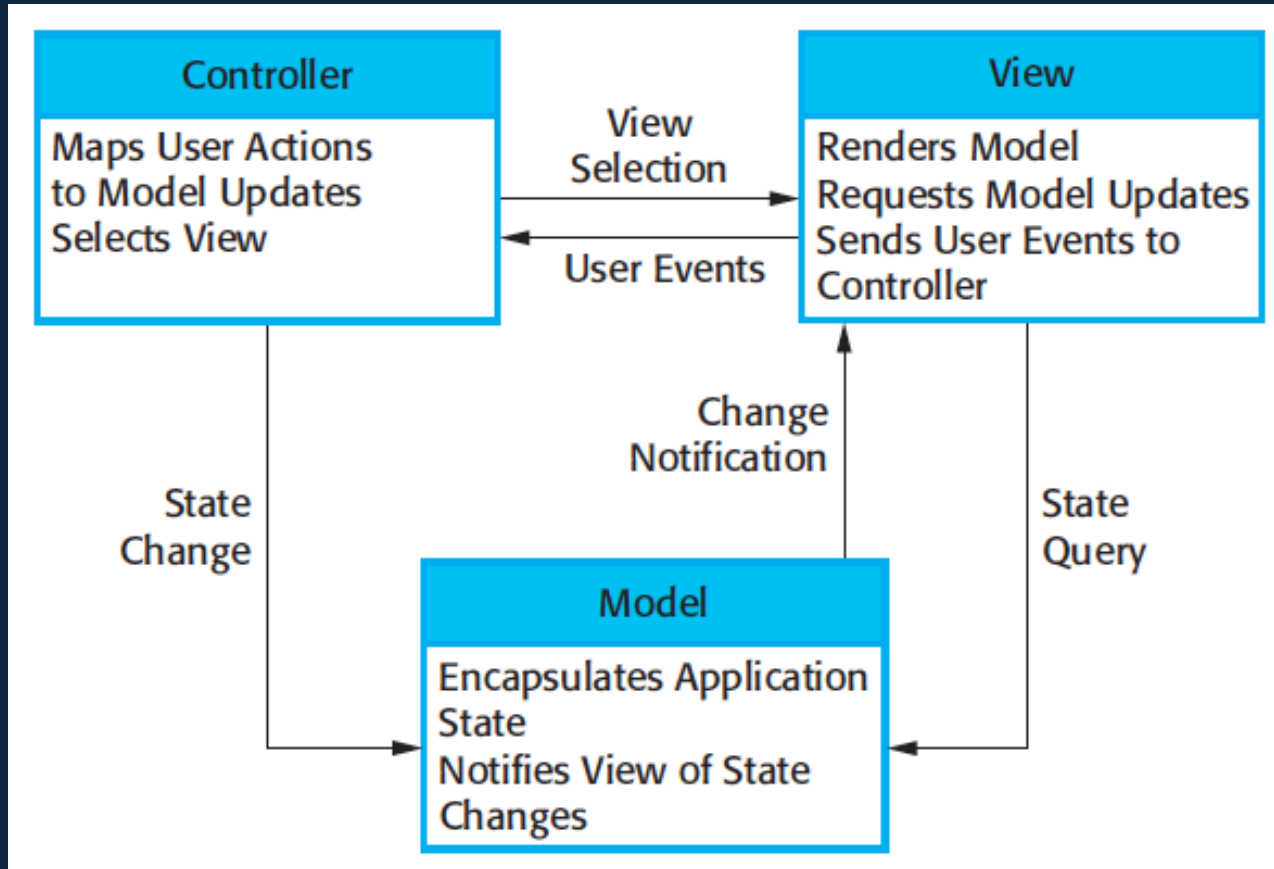
# Architectural Patterns

- Architectural Pattern – stylized, abstract description of good practice which have been tried and tested in different systems
- Some of the popular architectural patterns
  - MVC (Model-View-Controller) architecture
  - Layered architecture
  - Repository architecture
  - Client-Server architecture
  - Pipe and Filter

# MVC



# MVC

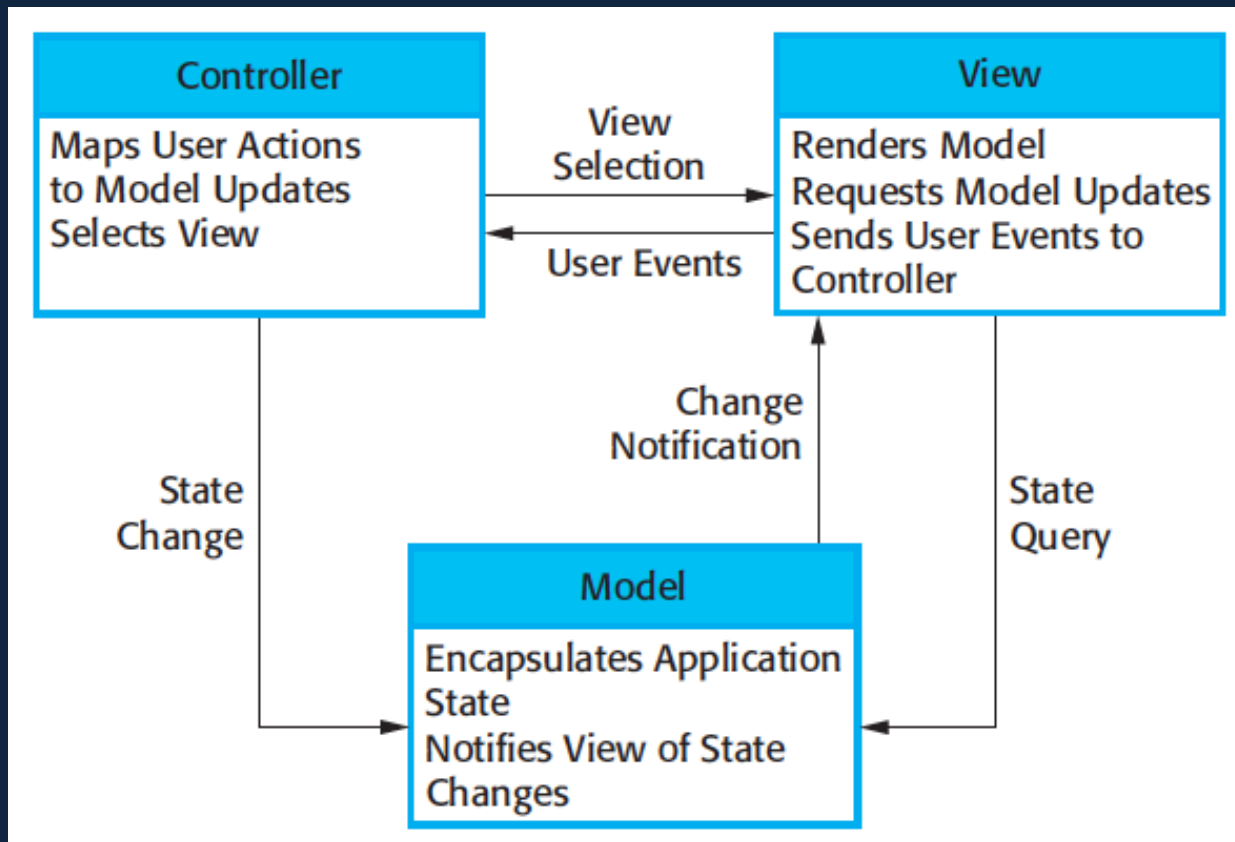


**Model** represents the data model, (e.g., data + business rules).

It manages the behavior and data of the application

# MVC

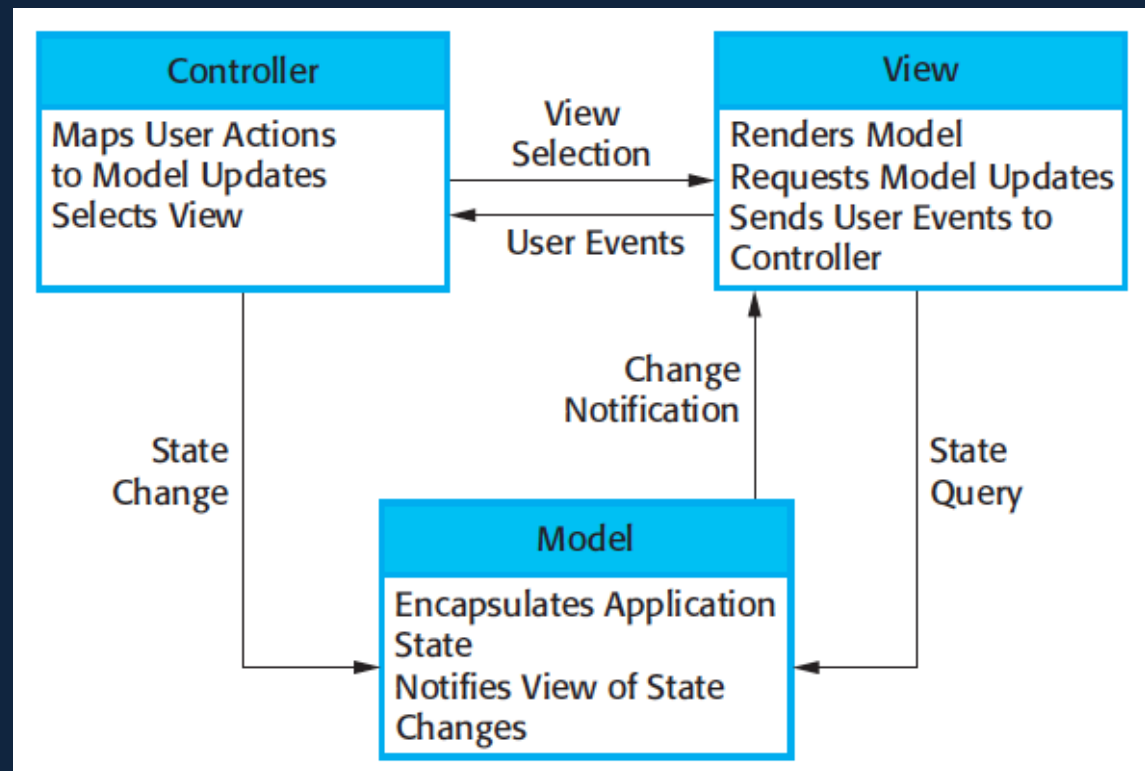
**View** represents the screens shown to the users. It manages the GUI and textual interface.



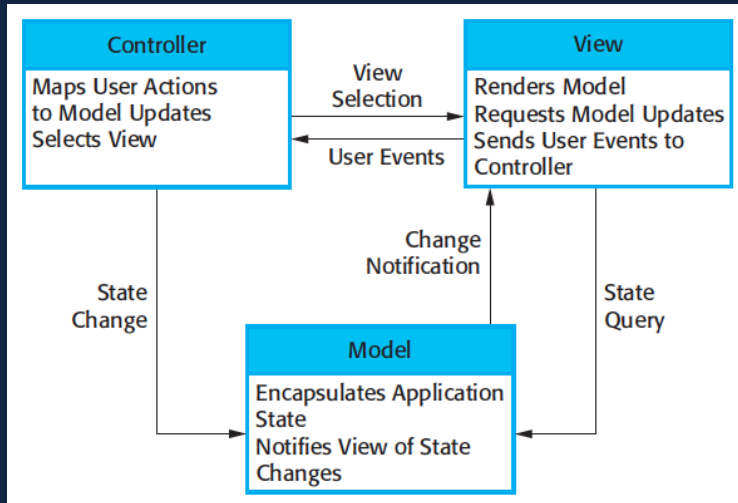


# MVC

**Controller** represents interactions from the users that change data and/or view. It interprets mouse clicks, keyboard inputs and commands the model or view to update accordingly.



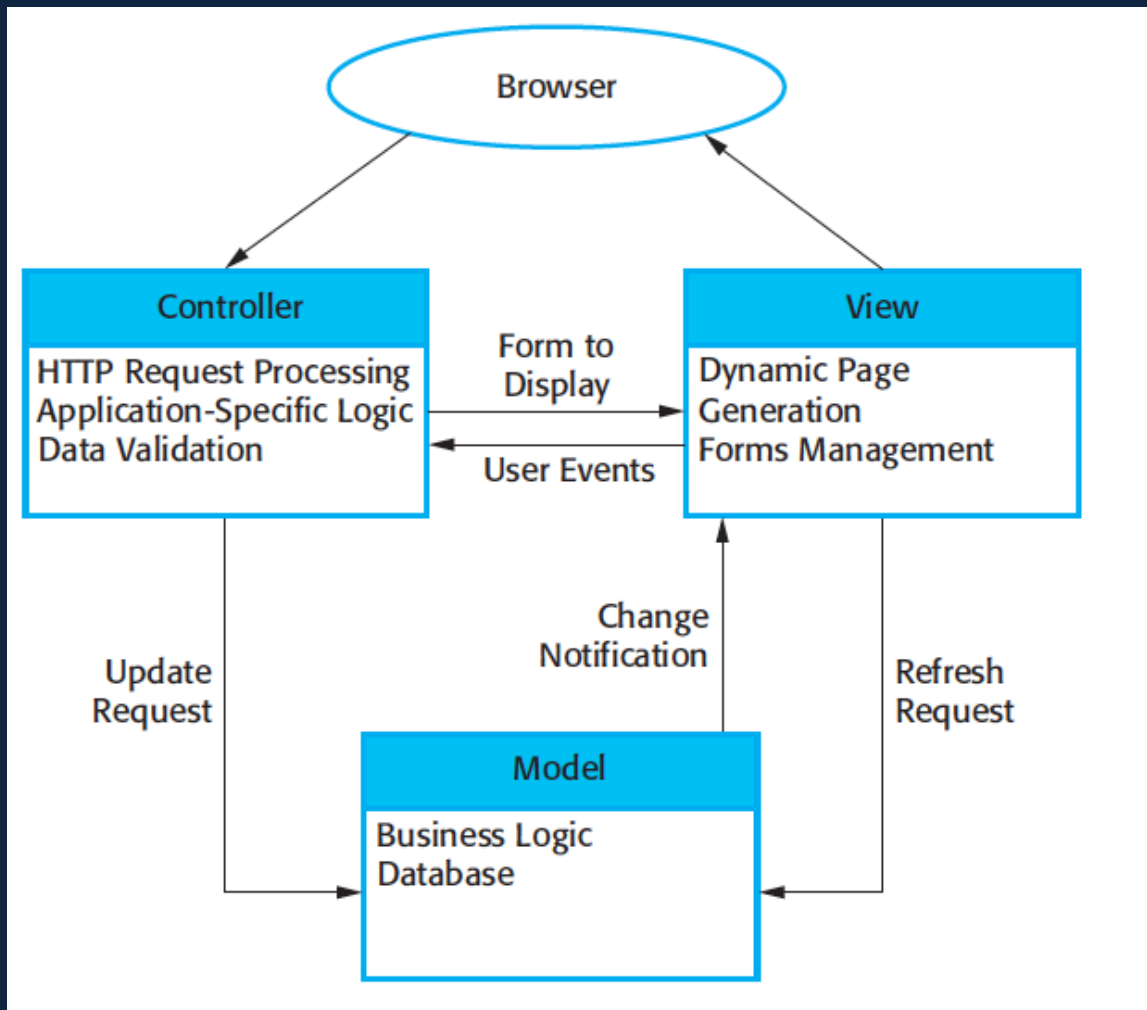
# MVC



## How it works:

1. VIEW displays UI
2. User inputs a command
3. CONTROLLER handles the input
4. CONTROLLER either updates the MODEL
  - a) Or Changes the VIEW selection
5. Updated MODEL notifies VIEW of changes
  - a) or Changed VIEW selection queries MODEL for data
6. VIEW renders the new UI

# MVC Example



# MVC

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. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
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.

# Layered

User Interface

User Interface Management  
Authentication and Authorization

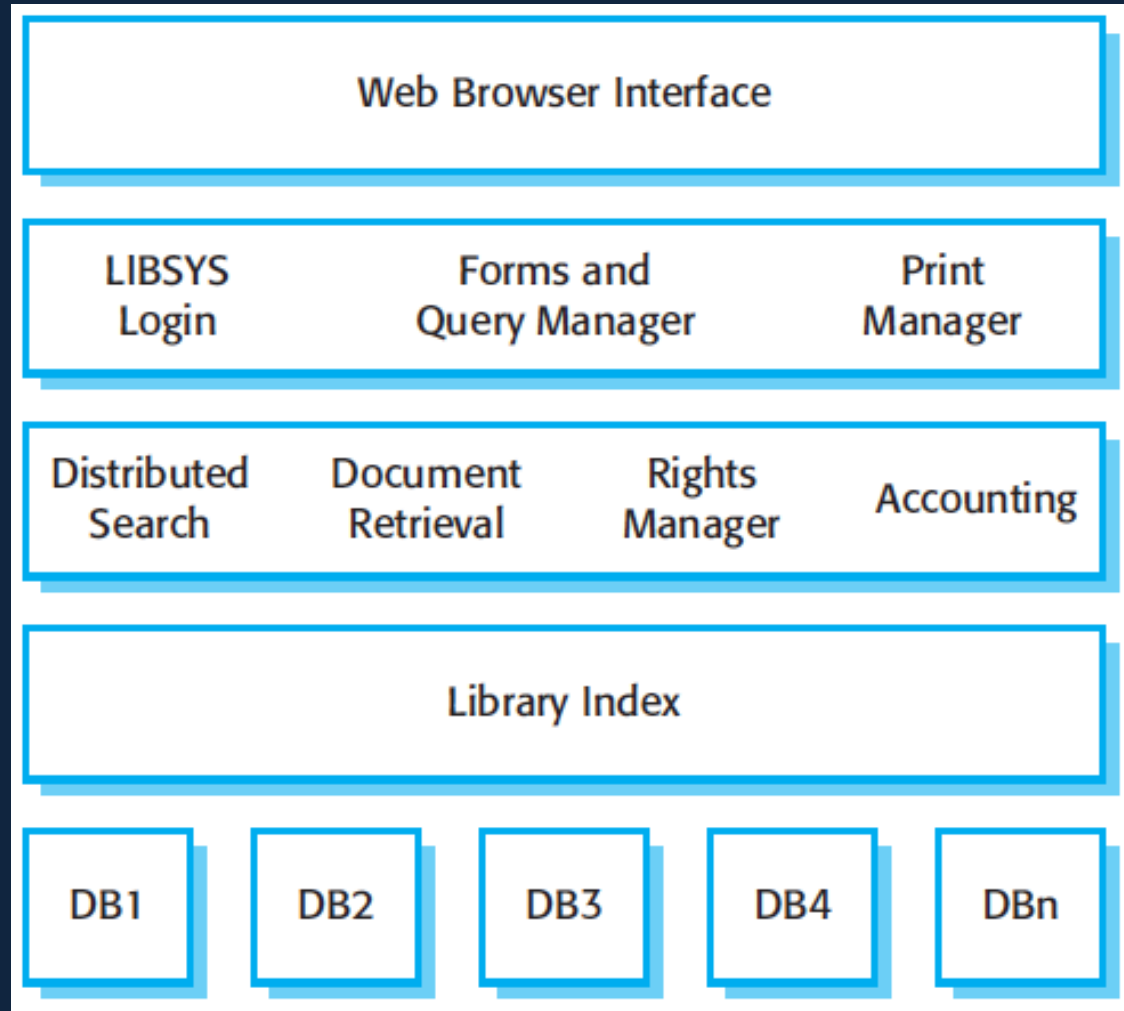
Core Business Logic/Application Functionality  
System Utilities

System Support (OS, Database etc.)

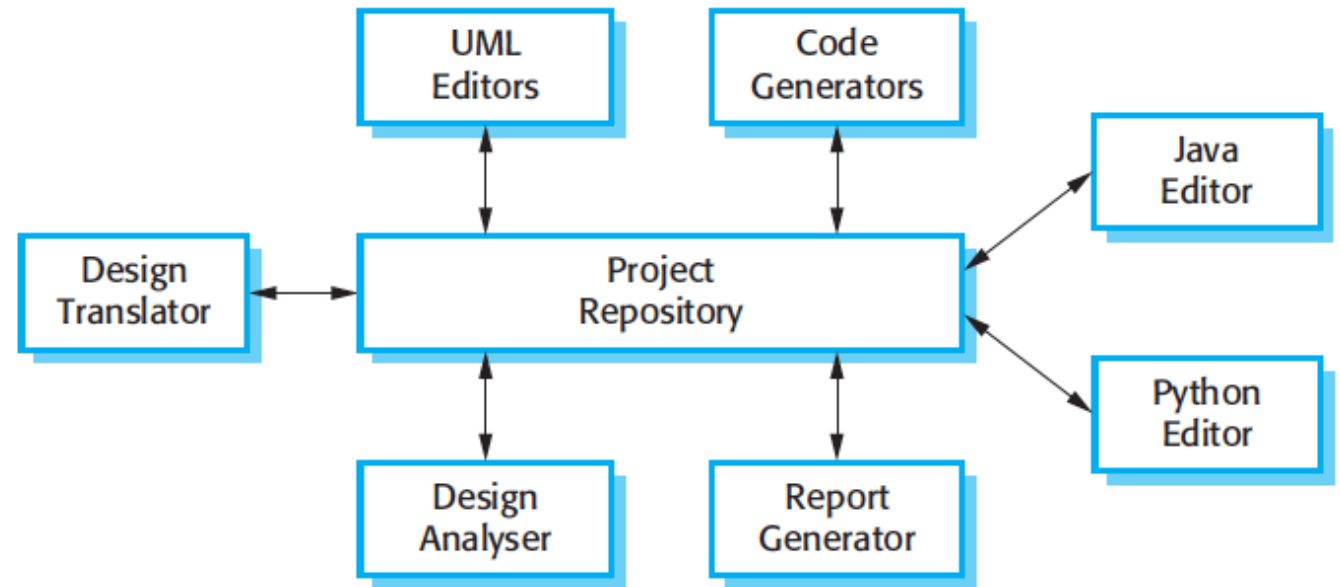
# Layered

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. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different libraries, as shown in Figure 6.7.
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.

# Layered



# Repository



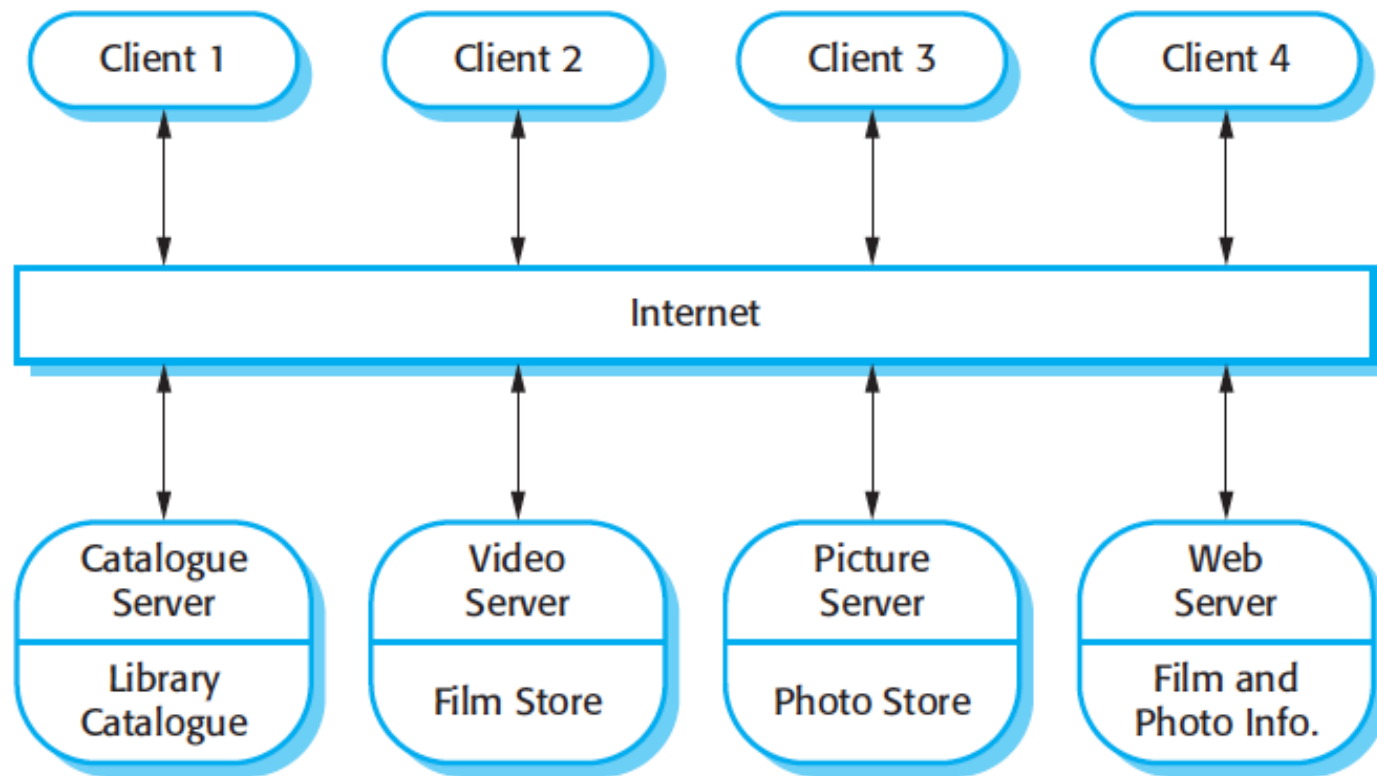
**Figure 6.9** A repository architecture for an IDE



# Repository

Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure 6.9 is an example of an IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

# Client-Server

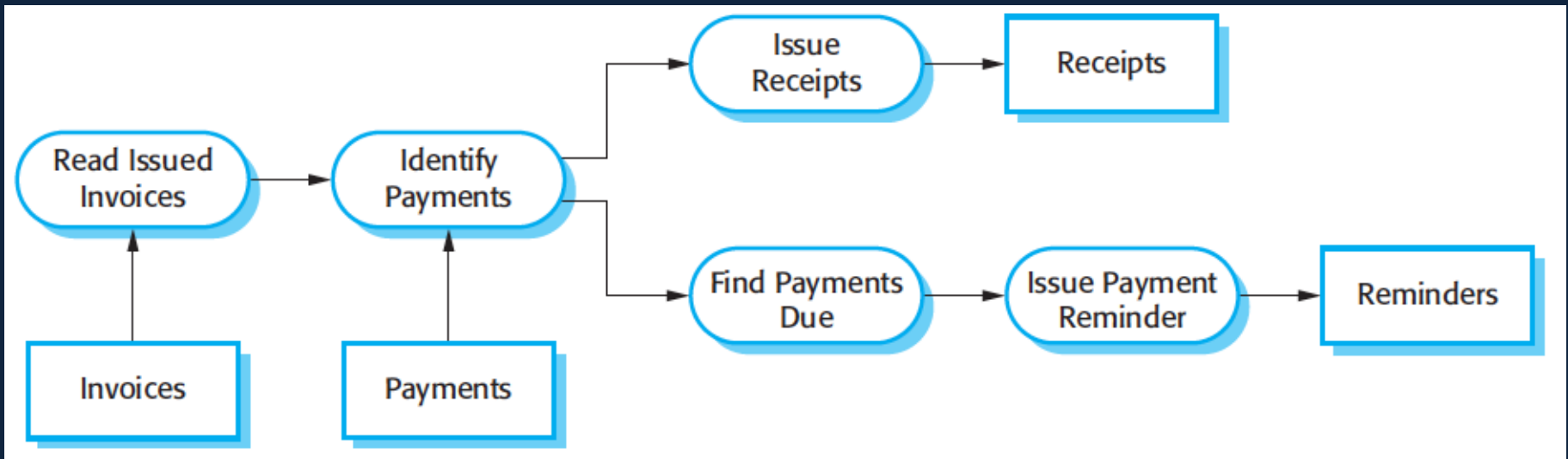


**Figure 6.11** A client—server architecture for a film library

# Client-Server

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.
Example	Figure 6.11 is an example of a film and video/DVD library organized as a client-server system.
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.

# Pipe and Filter



# Pipe and Filter

Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Figure 6.13 is an example of a pipe and filter system used for processing invoices.
When used	Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.

# Summary

- A software architecture is a description of how a software system is organized.
- Several properties of a system are affected by the architecture used, including performance, robustness, distributability and maintainability
- Architectural patterns are means of reusing knowledge about generic system architectures.
- Commonly used architectural patterns include Model-View-Controller, Layered Architecture, Repository, Client-Server, and Pipe and Filter.

# Reference

- I. Sommerville, *Introduction to Software Engineering 9<sup>th</sup> ed.*  
Addison-Wesley, 2011