جامعة الملك سعود
King Saud University
College of Computer & Information Sciences

Architectural Patterns

Chapter 6 (Sommerville)

Fall Semester 2021
1st Semester 1443 H

Topics

Architectural Patterns

1. MVC Architecture
2. Layered Architecture
3. Repository Architecture
4. Client-Server Architecture
5. Pipe and Filter Architecture

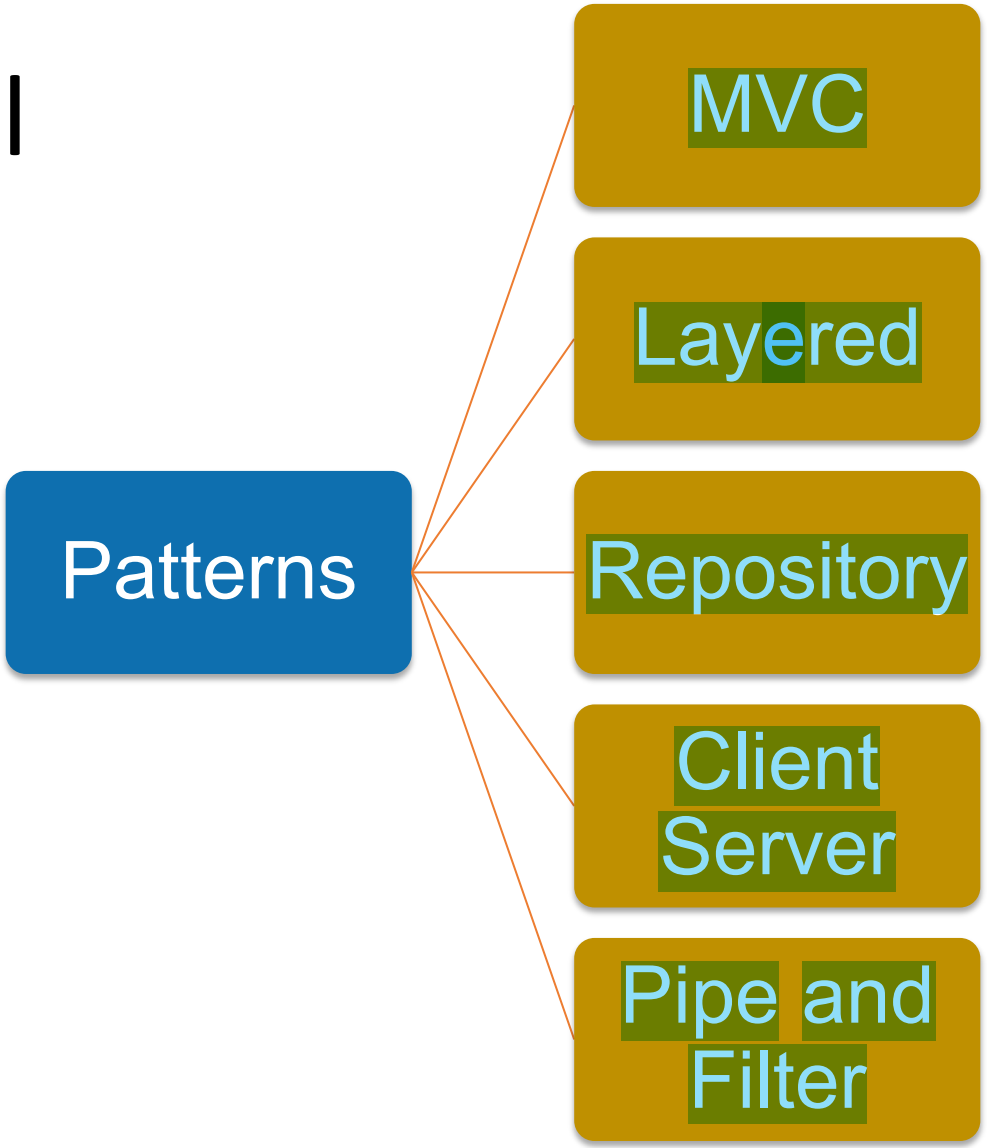
Software Specification (Requirements Engineering)	Software Design & Development	Software Validation	Software Evolution
Customers and engineers **define** the software that is to be produced and the **constraints** on its operation.	The software is **designed** and **programmed**.	The software is **checked** to ensure that it is what the customer requires.	The software is **modified** to reflect changing customer and market requirements

The 4 Fundamental SW Process Activities

Architectural Patterns

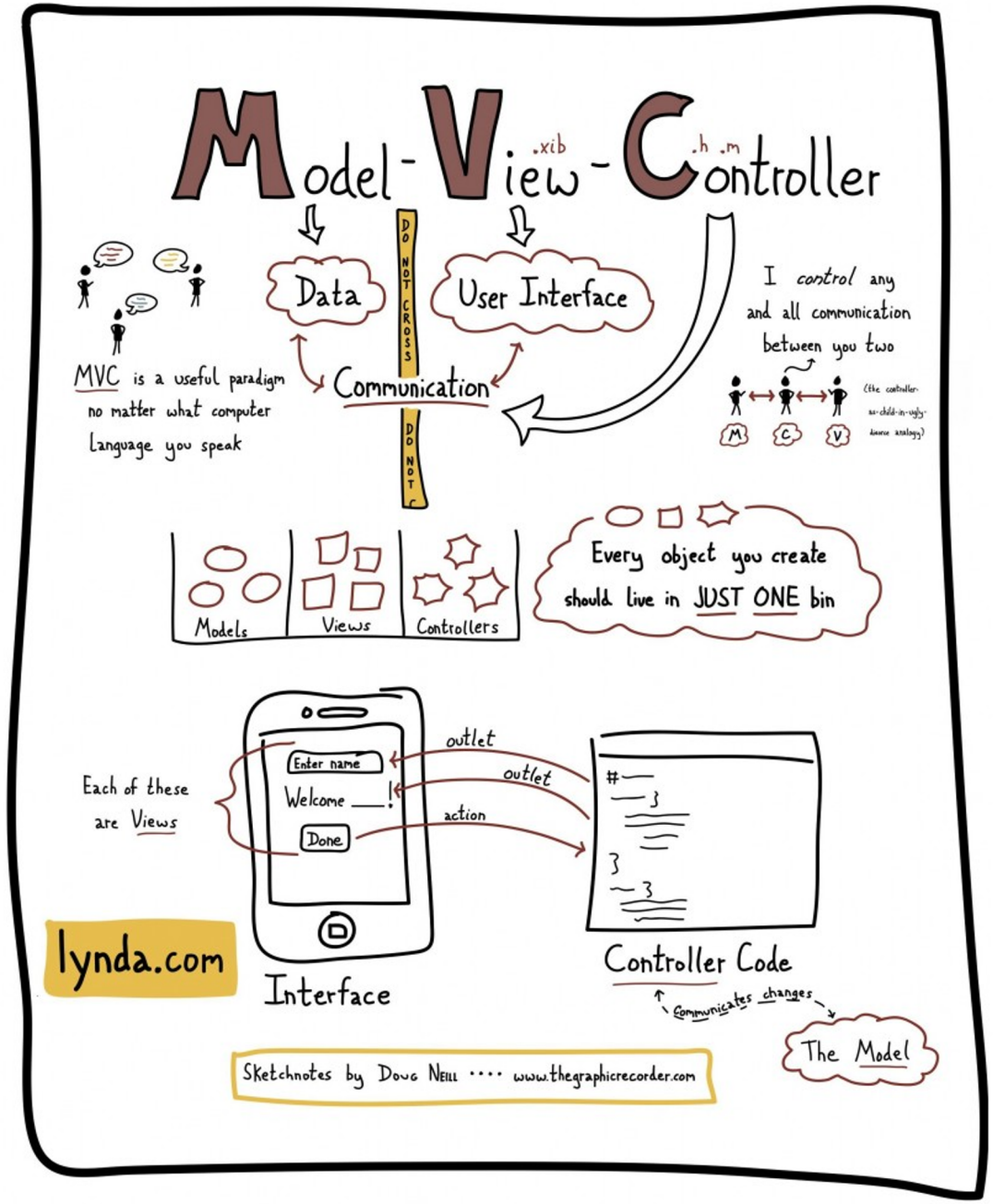
- Patterns are a means of **representing**, **sharing** and **reusing** knowledge.
- An architectural pattern is a **stylized description** of good design practice, *which has been tried and tested in different environments*.
- Patterns should include information about **when they are** and **when the are not useful**.
- Patterns may be represented using *tabular* and *graphical* descriptions.

Architectural Patterns



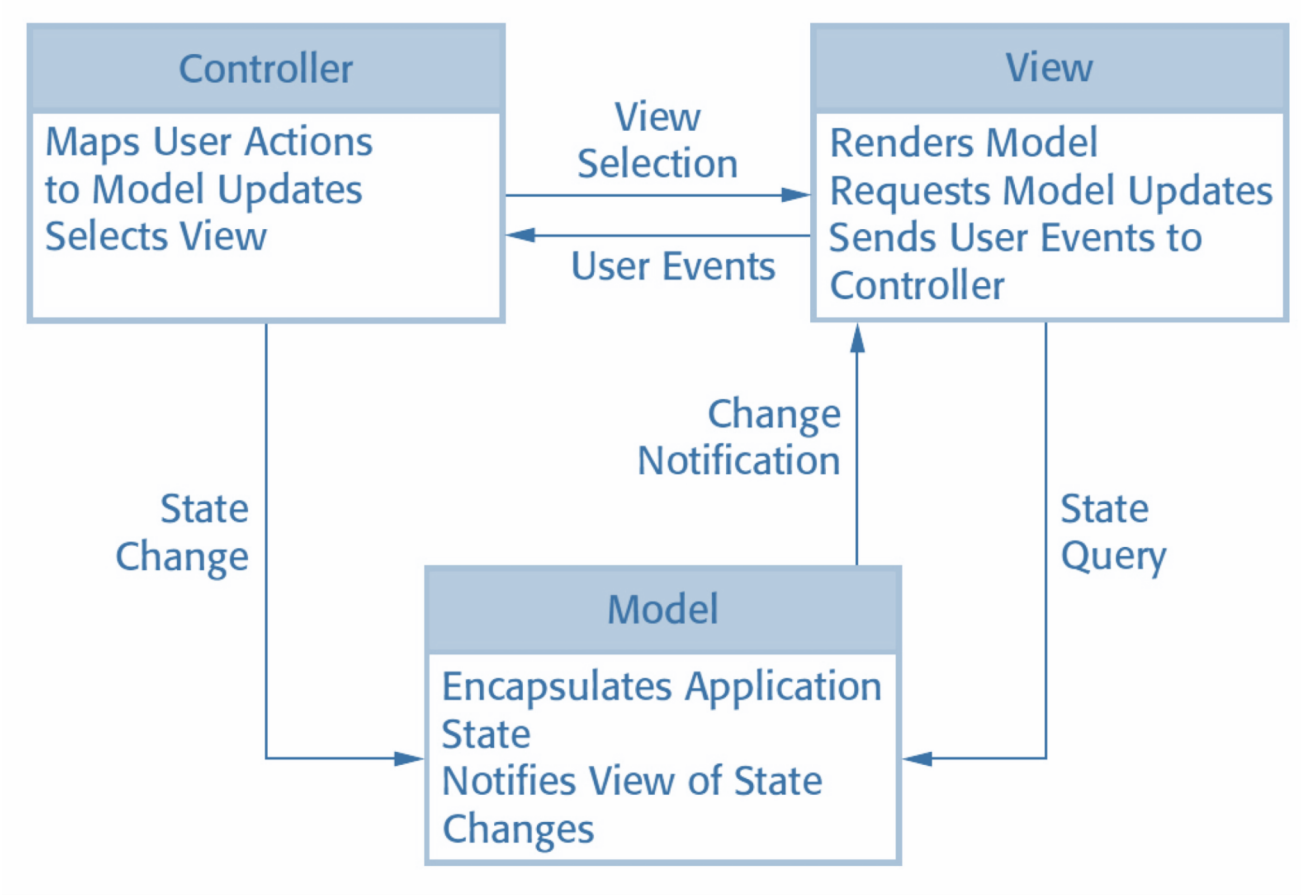
1-MVC Architecture

Model-View-Controller



The Model-View-Controller (MVC) pattern

College of Computer and Information Sciences

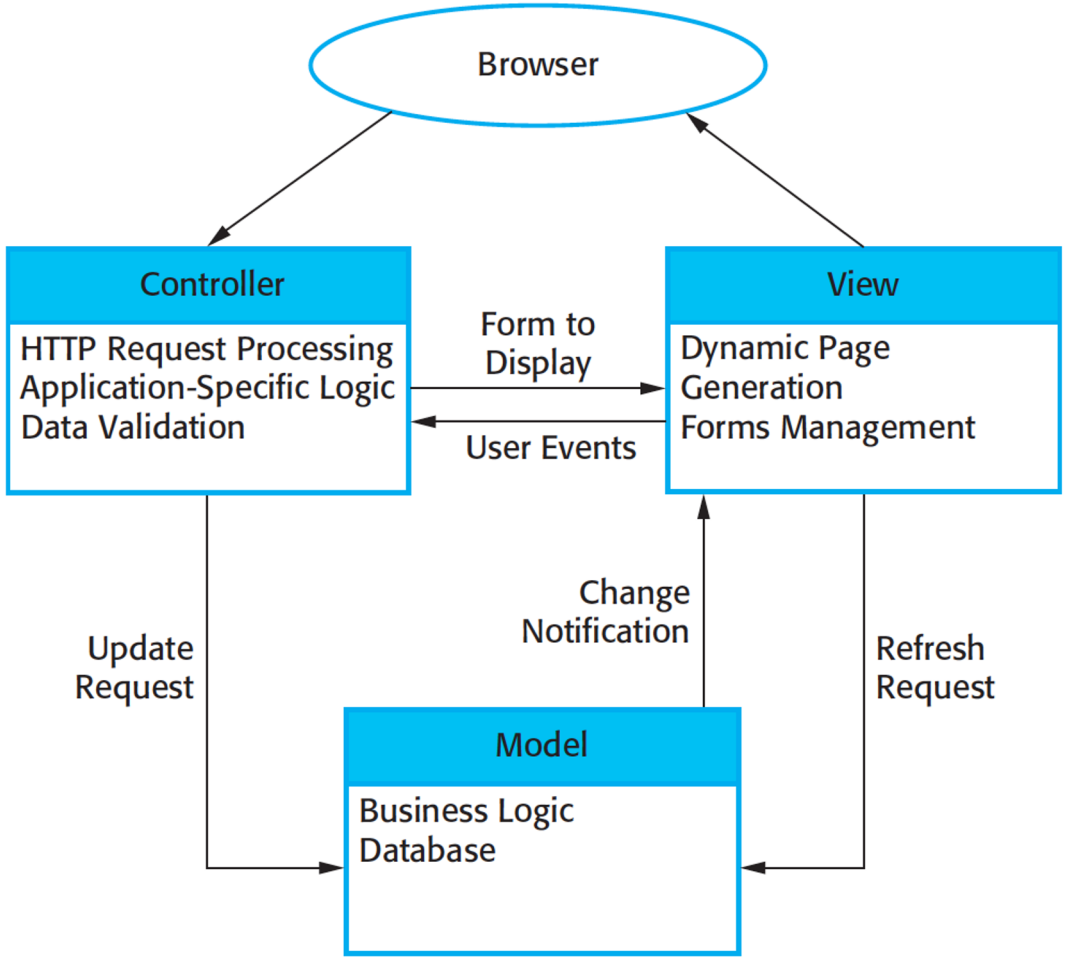


The Model-View-Controller (MVC) pattern

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.

The Model-View-Controller (MVC) pattern

Web Application Architecture using the MVC Pattern



MVC Example

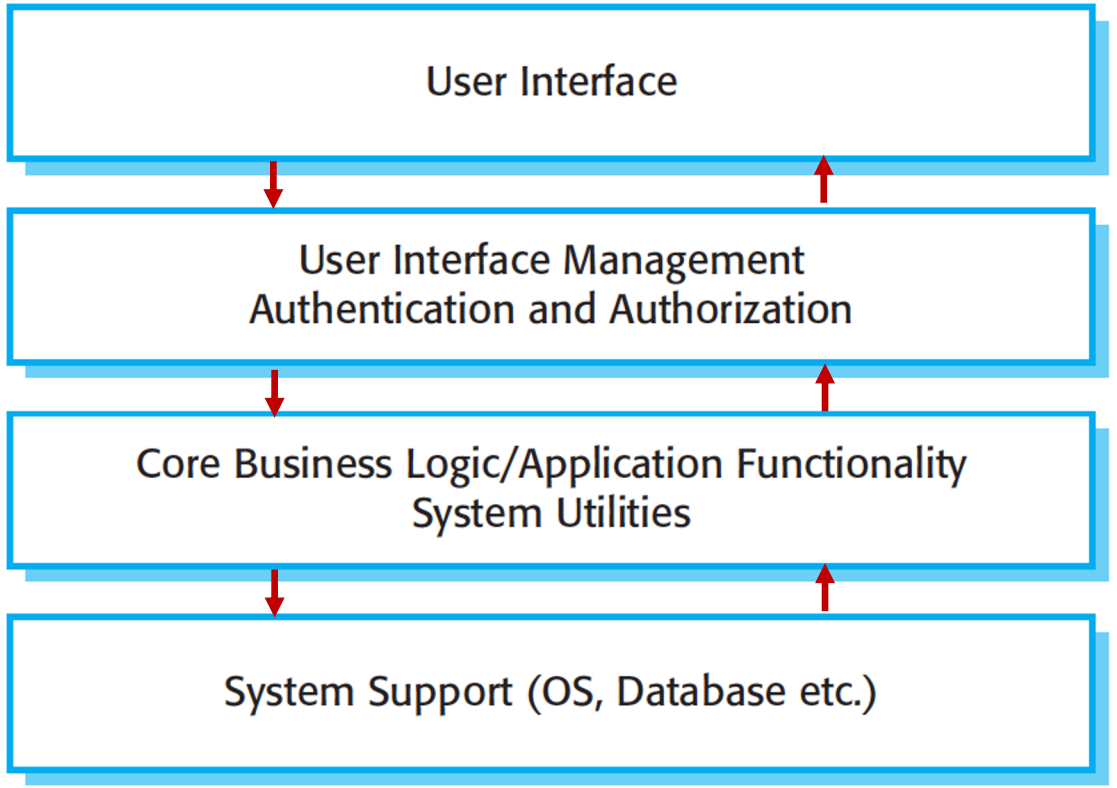
View= You
Waiter= Controller
Cook= Model
Refrigerator= Data

https://www.guru99.com/mvc-tutorial.html

2- Layered Architecture

Layered Architecture

- Used to model the interfacing of sub-systems.
- Organizes the system into layers each provides a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.



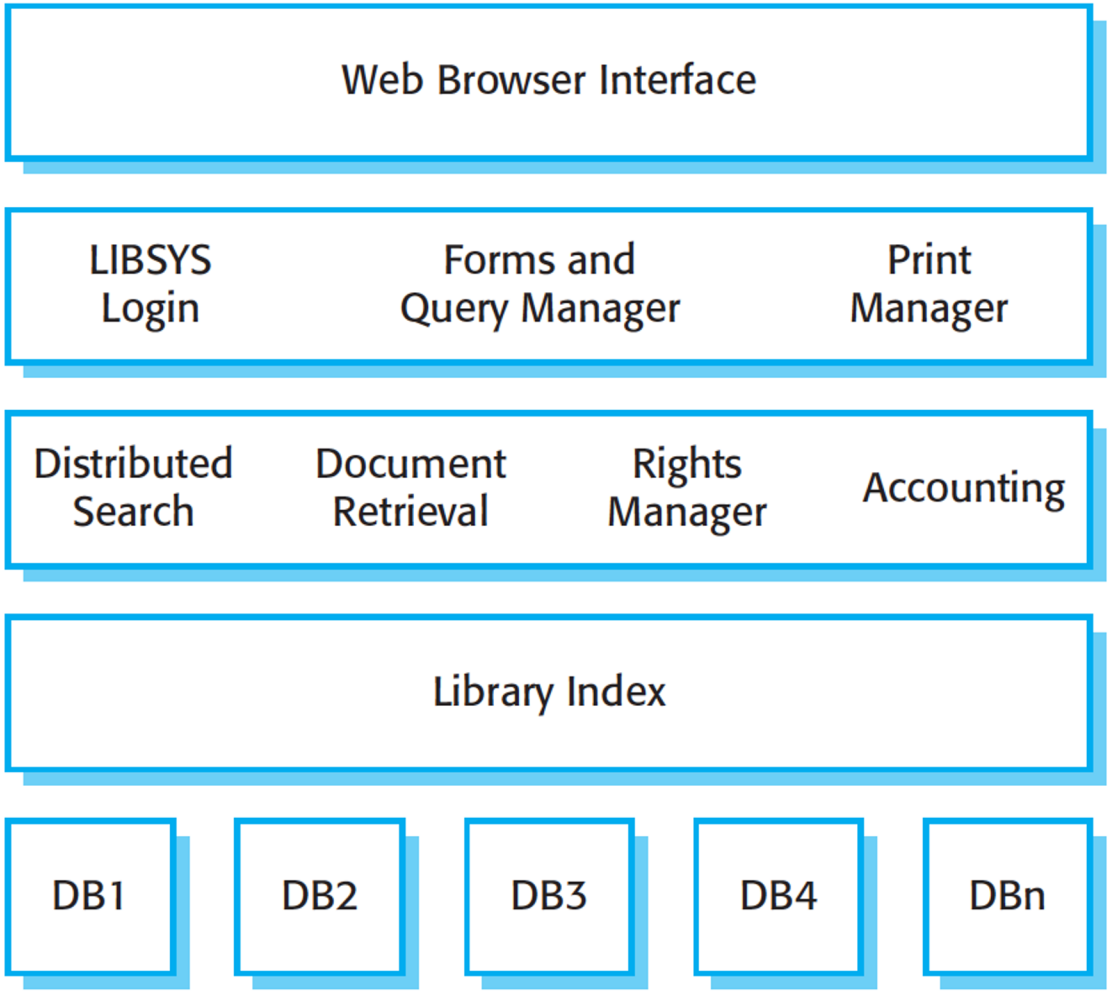
A Generic Layered Architecture

Layered Architecture

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 Architecture

The Architecture of the LIBSYS system



3-Repository Architecture- *Data Centered*

Repository Architecture

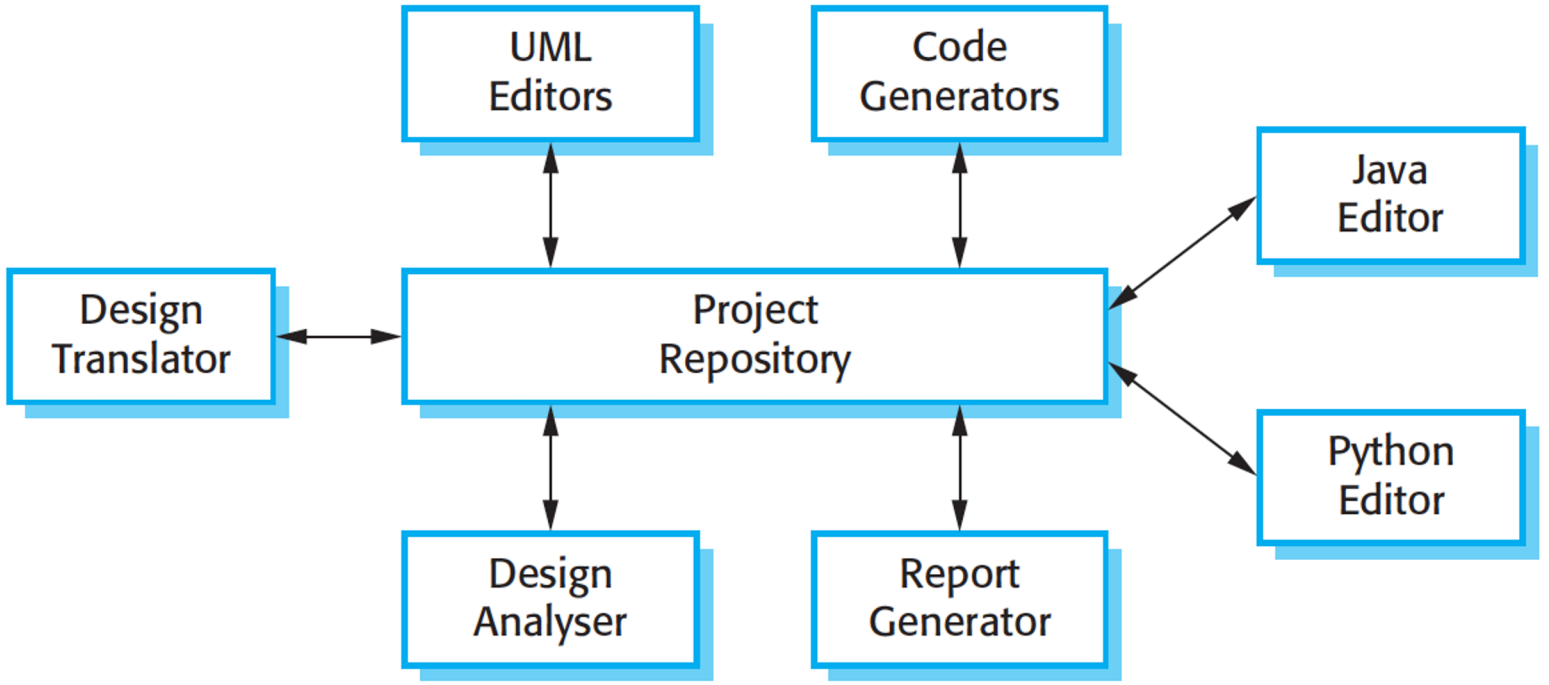
- Sub-systems must exchange data. This may be done in two ways:
  1. Shared data is held in a **central database or repository** and may be accessed by all sub-systems.
  2. Each sub-system **maintains its own database** and passes data explicitly to other sub-systems.

- When large amounts of **data are to be shared**, the repository model of sharing data is most commonly used as it is an **efficient data sharing** mechanism.

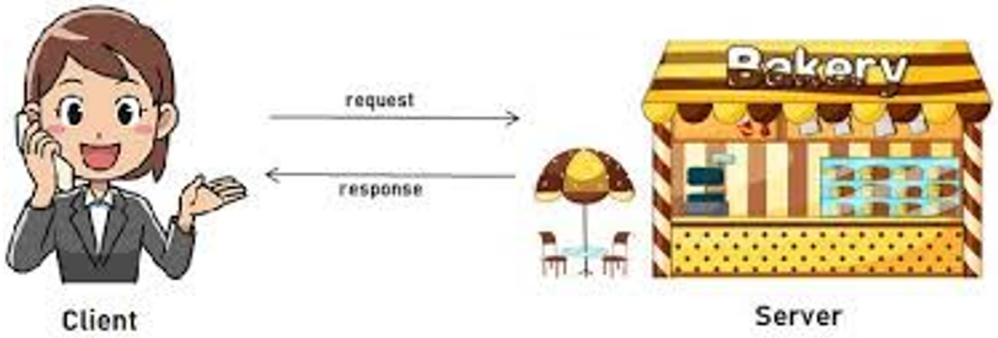
Repository Architecture

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.

Repository Architecture



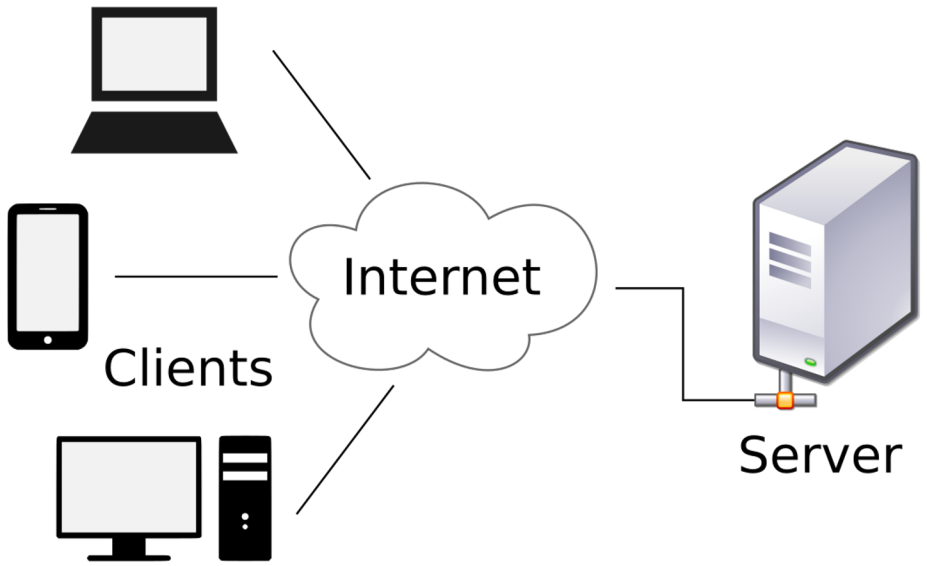
A repository Architecture for an IDE



4-Client-Server Architecture

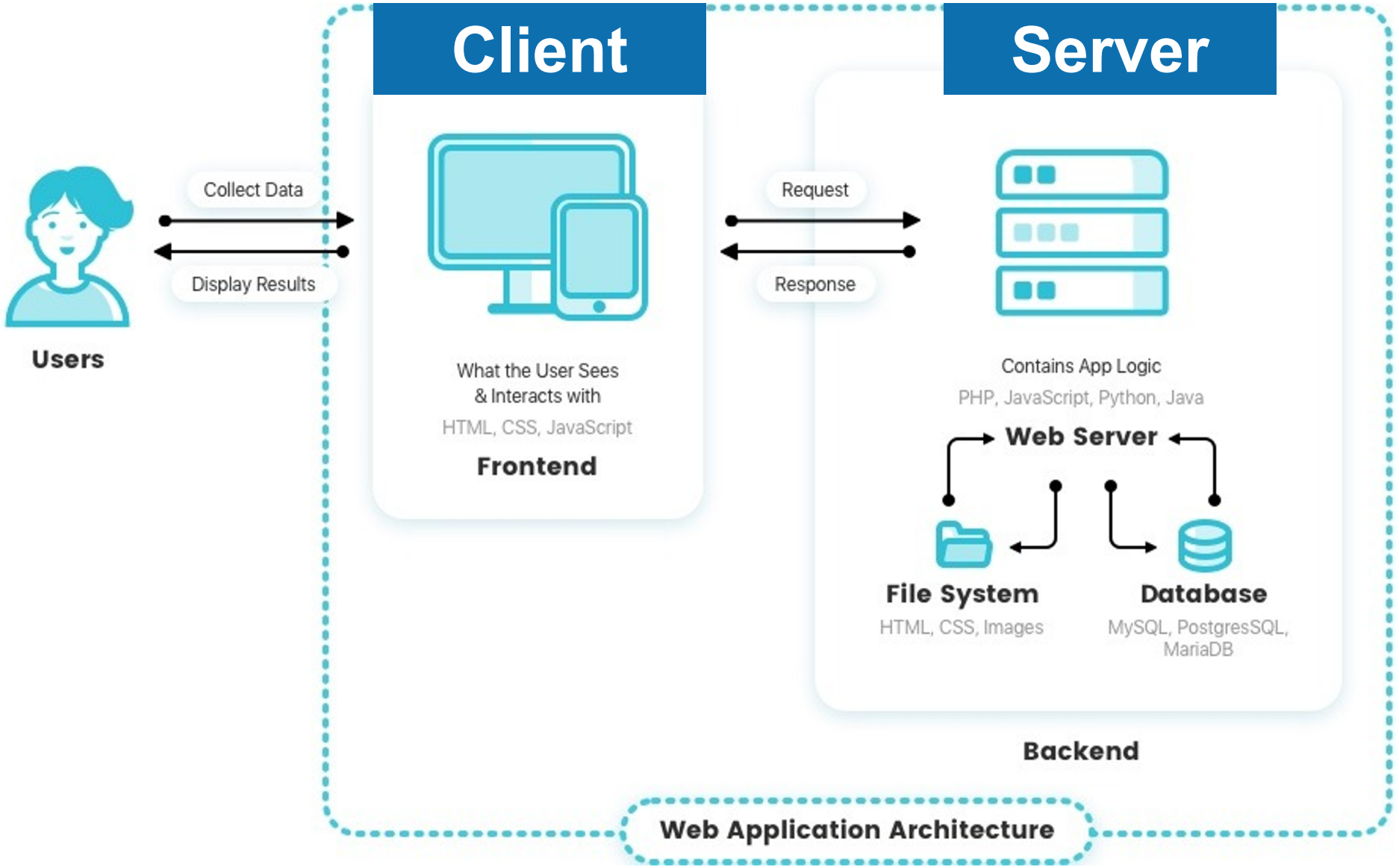
Client-Server Architecture

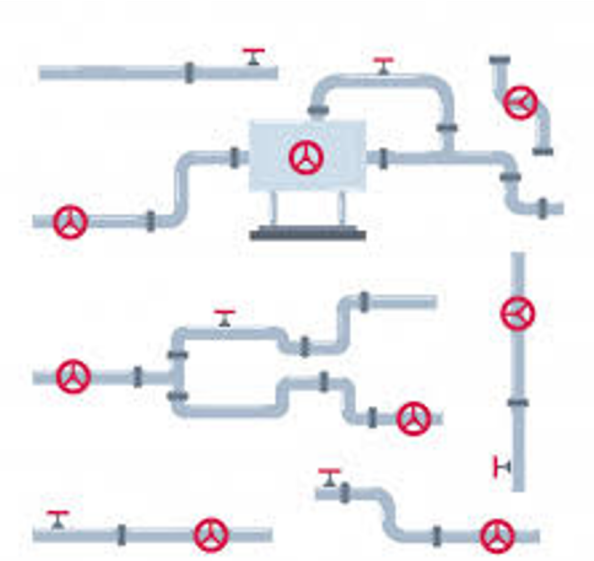
- This pattern consists of three parts:
  1. A server.
  2. Multiple clients.
  3. Network.
- The **server** ***provides services*** to the client.
- **Clients** ***request services*** from the server.
- Used for most online applications such as email, banking, and document sharing.



Client Server Architecture

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.


Client
Server
Collect Data
Display Results
Request
Response
Users
What the User Sees
& Interacts with
HTML, CSS, JavaScript
Frontend
Contains App Logic
PHP, JavaScript, Python, Java
Web Server
File System
HTML, CSS, Images
Database
MySQL, PostgresSQL,
MariaDB
Backend
Web Application Architecture



5- Pipe and Filter Architecture

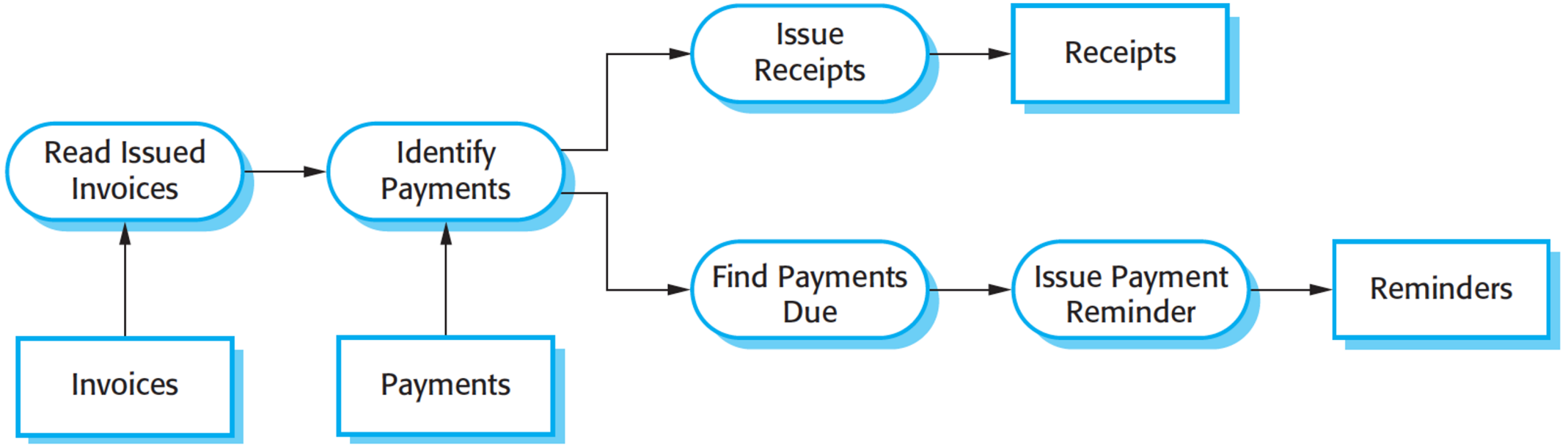
Pipe and Filter Architecture

- Functional transformations process their inputs to produce outputs.
- Variants of this approach are very common. When transformations are *sequential*, this is a **batch sequential model** which is extensively used in data processing systems.
- Not really suitable for interactive systems.

Pipe and Filter Architecture

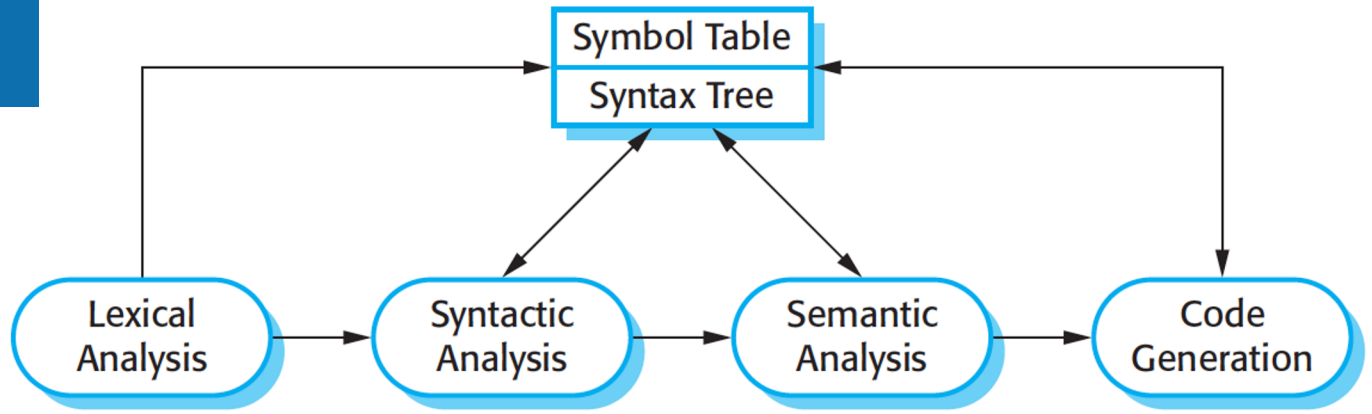
Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Figure 6.13 is an example of 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.

Pipe and Filter Architecture



An example of the pipe and filter architecture

Compiler Architecture



Lexical analyzer	Takes input language tokens and converts them to an internal form.
Symbol table	Holds information about the names of entities (variables, class names, object names, etc.) used in the text that is being translated.
Syntax analyzer	Checks the syntax of the language being translated. It uses a defined grammar of the language and builds a syntax tree.
Syntax tree	Internal structure representing the program being compiled.
Semantic analyzer	Uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
Code generator	Walks the syntax tree and generates abstract machine code.

Summary

- Software architectures are essential to develop and maintain large-scale, long-living software systems.
- The understanding of these systems is improved by a high- level abstract view on a system.
- Architecture supports the reuse of components and frameworks.
- Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used, and discuss its advantages and disadvantages.
- Commonly used architectural patterns include Model-View-Controller, Layered Architecture, Repository, Client–server, and Pipe and Filter.

Thank you!