

Software Engineering

Prepared by Dr. Yasser Abdelhamid

IT 382
Software Engineering

Lecture 9



LECTURE OUTLINE

- Architectural Design
- Software Architecture
- Application Architecture

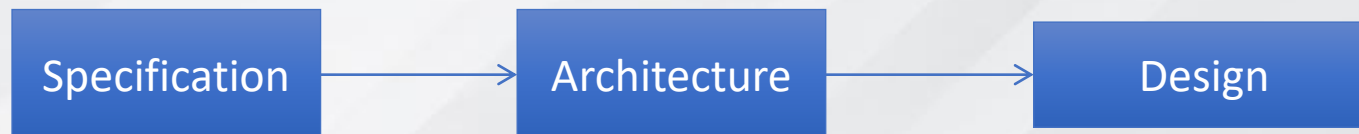
SOFTWARE ARCHITECTURE

- Architectural design is identifying
 - The sub-systems
 - Sub-system control
 - Sub-system communication

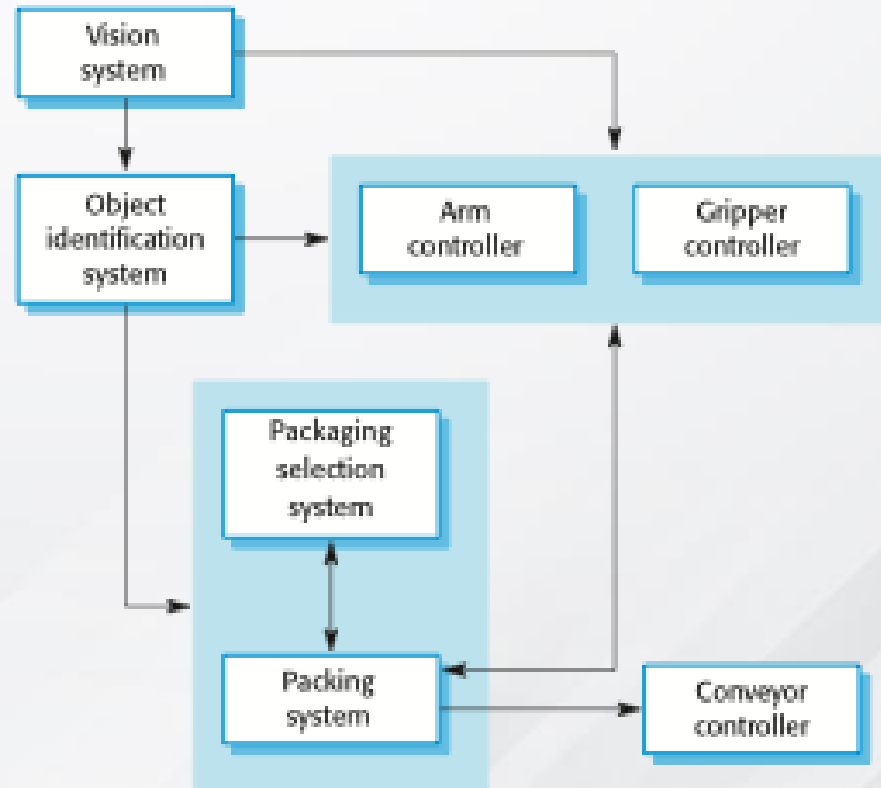


ARCHITECTURAL DESIGN

1. An early stage of the system design process.
2. Represents the link between specification and design processes.
3. Often carried out in parallel with some specification activities.



THE ARCHITECTURE OF A PACKING ROBOT CONTROL SYSTEM



ARCHITECTURAL 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 (modules)** .

ARCHITECTURAL ABSTRACTION

- **Architecture in the large** is concerned with:
 - The architecture of **complex enterprise systems** that include *other systems, programs, and program components*.
 - These complex enterprise systems
 - Are distributed over **different computers**,
 - May be owned and managed by **different companies**

ADVANTAGES OF EXPLICIT ARCHITECTURE

- **Stakeholder communication**
 - Architecture may be used as a **focus of discussion** by system stakeholders.
- **System analysis**
 - More **system understanding** and **easier management**
- **Large-scale reuse**
 - The architecture may be **reusable** across a range of systems

ARCHITECTURAL REPRESENTATIONS

- Simple, **informal block diagrams** showing:
 - Entities and relationships are the most frequently used method for documenting software architectures.
- But these have been criticized because
 - **They lack semantics**, do not show the types of relationships between entities nor the visible properties of entities in the architecture.

BOX AND LINE DIAGRAMS ... TRADE OFF

- - **Negative:**
 - Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- + **Positive**
 - Useful for communication with stakeholders and for project planning.

USE OF ARCHITECTURAL MODELS

- 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.
- 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 DECISIONS

- Architectural design **is a creative process**
- Examples of architectural design decisions
 - Is there a **generic application architecture** that can be used?
 - How will the system be **distributed**?
 - What **architectural styles** are appropriate?
 - 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**?

ARCHITECTURE REUSE

- Systems in the same domain often have **similar architectures** that reflect domain concepts.
- Application product lines are built around a **core architecture** with variants that satisfy particular customer requirements.
- The architecture of a system may be designed around **one or more** architectural patterns or 'styles'.

ARCHITECTURE AND SYSTEM CHARACTERISTICS (NON-FUNCTIONAL REQUIREMENTS)

- **Performance**
 - minimize communications.
 - Use large rather than fine-grain components.
- **Security**
 - Use a layered architecture with critical assets in the inner layers.
- **Safety**
 - Localize safety-critical features in a small number of sub-systems.
- **Availability**
 - Include redundant components and mechanisms for fault tolerance.
- **Maintainability**
 - Use fine-grain, replaceable components.

ARCHITECTURAL VIEWS

- We have 2 what's:
 - What **views** or perspectives are useful when designing and documenting a system's architecture?
 - What **notations** should be used for describing architectural models?

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**
 - How 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.

4 + 1 VIEW MODEL OF SOFTWARE ARCHITECTURE

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

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 they are not useful.
- Patterns may be represented using **tabular** and **graphical** descriptions.

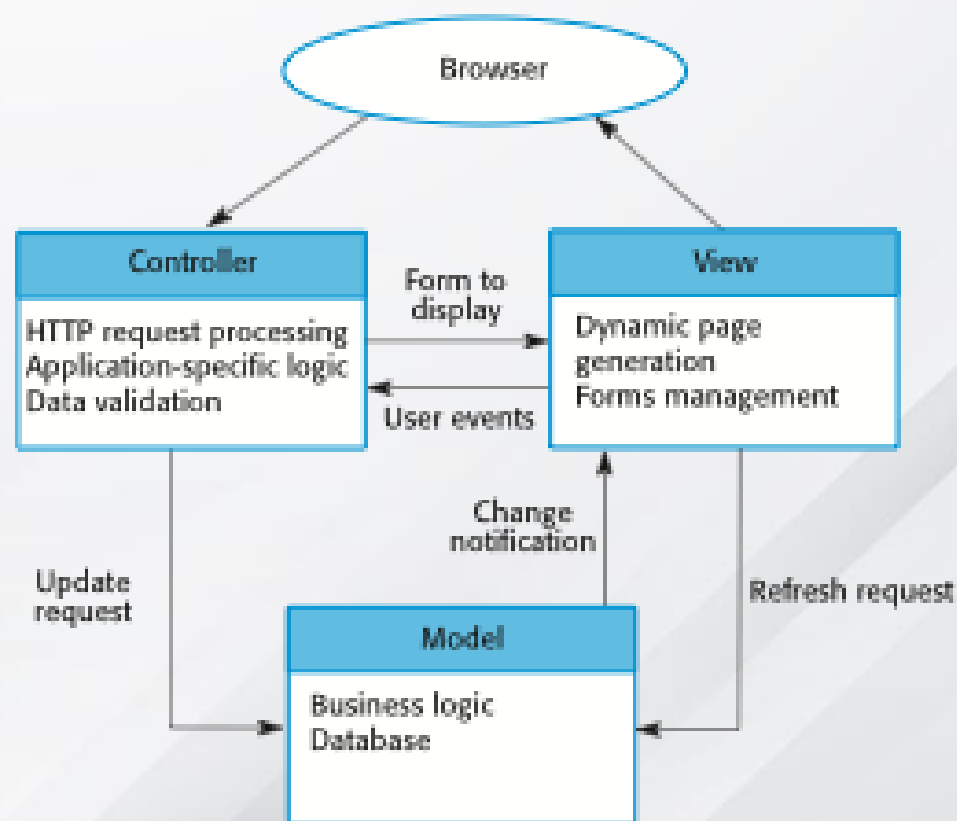
THE MODEL-VIEW-CONTROLLER (MVC) PATTERN

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.
Example	A web-based application system organized using the MVC pattern.
When used	1. Used when there are multiple ways to view and interact with data. 2. Used when the future requirements for interaction and presentation of data are unknown.
Advantages	1. Allows the data to change independently of its representation and vice versa. 2. 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

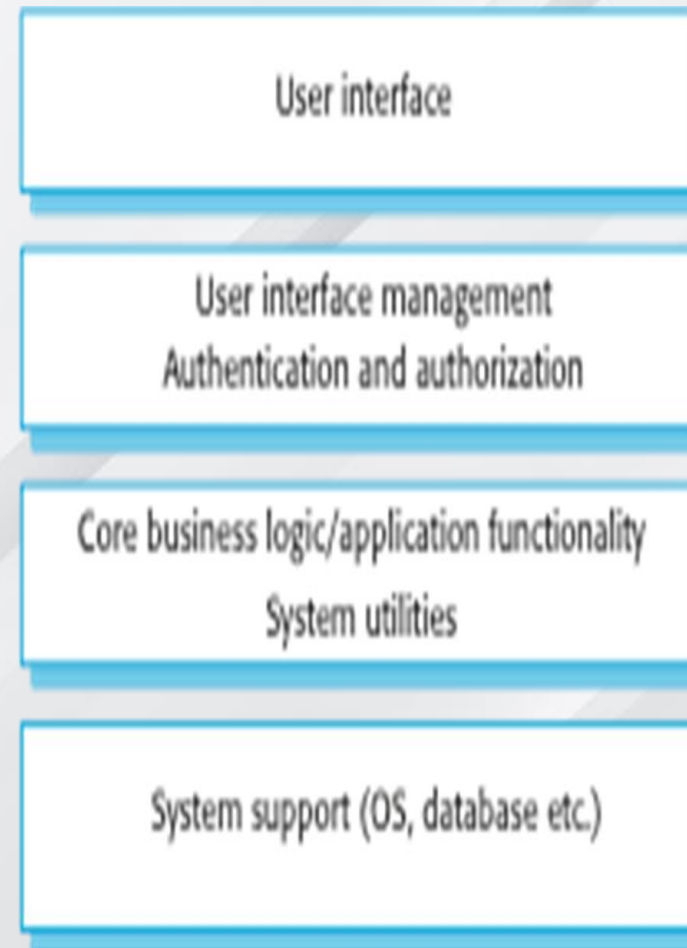
Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data . The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines 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.
Example	A web-based application system organized using the MVC pattern.
When used	1. Used when there are multiple ways to view and interact with data. 2. Used when the future requirements for interaction and presentation of data are unknown.
Advantages	1. Allows the data to change independently of its representation and vice versa. 2. 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.

WEB APPLICATION ARCHITECTURE USING THE MVC PATTERN

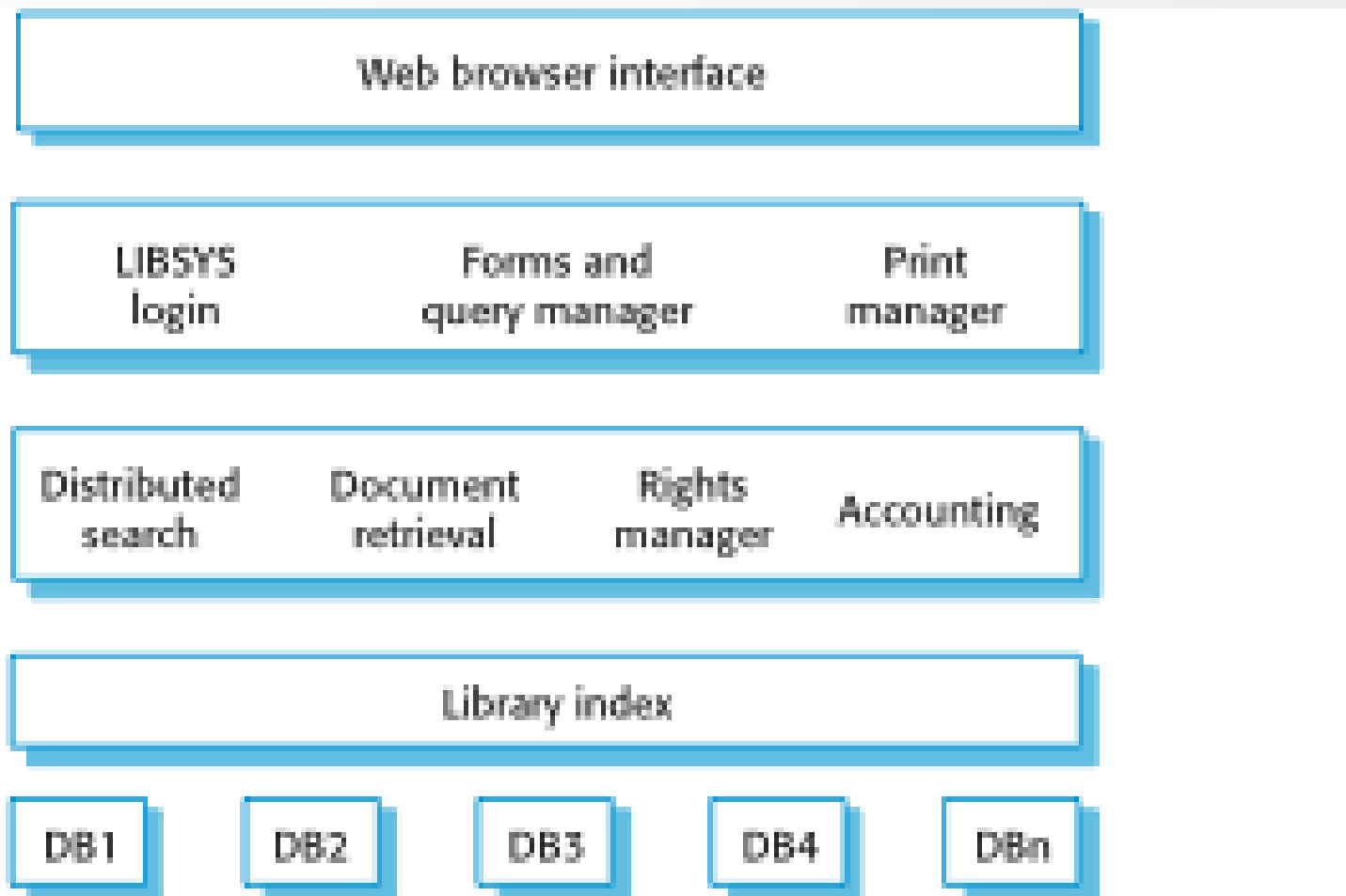


LAYERED ARCHITECTURE

- Organises the system into a set of layers each of which provide 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.



THE ARCHITECTURE OF THE LIBSYS SYSTEM



THE LAYERED ARCHITECTURE PATTERN

Lo. 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.
Example	A layered model of a system for sharing copyright documents held in different libraries.
When used	1. Used when building new facilities on top of existing systems; 2. When the development is spread across several teams with each team responsibility for a layer of functionality; 3. When there is a requirement for multi-level security.
Advantages	1. Allows replacement of entire layers as long as the interface is maintained. 2. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	1. In practice, providing a clean separation between layers is often difficult. 2. A high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. 3. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

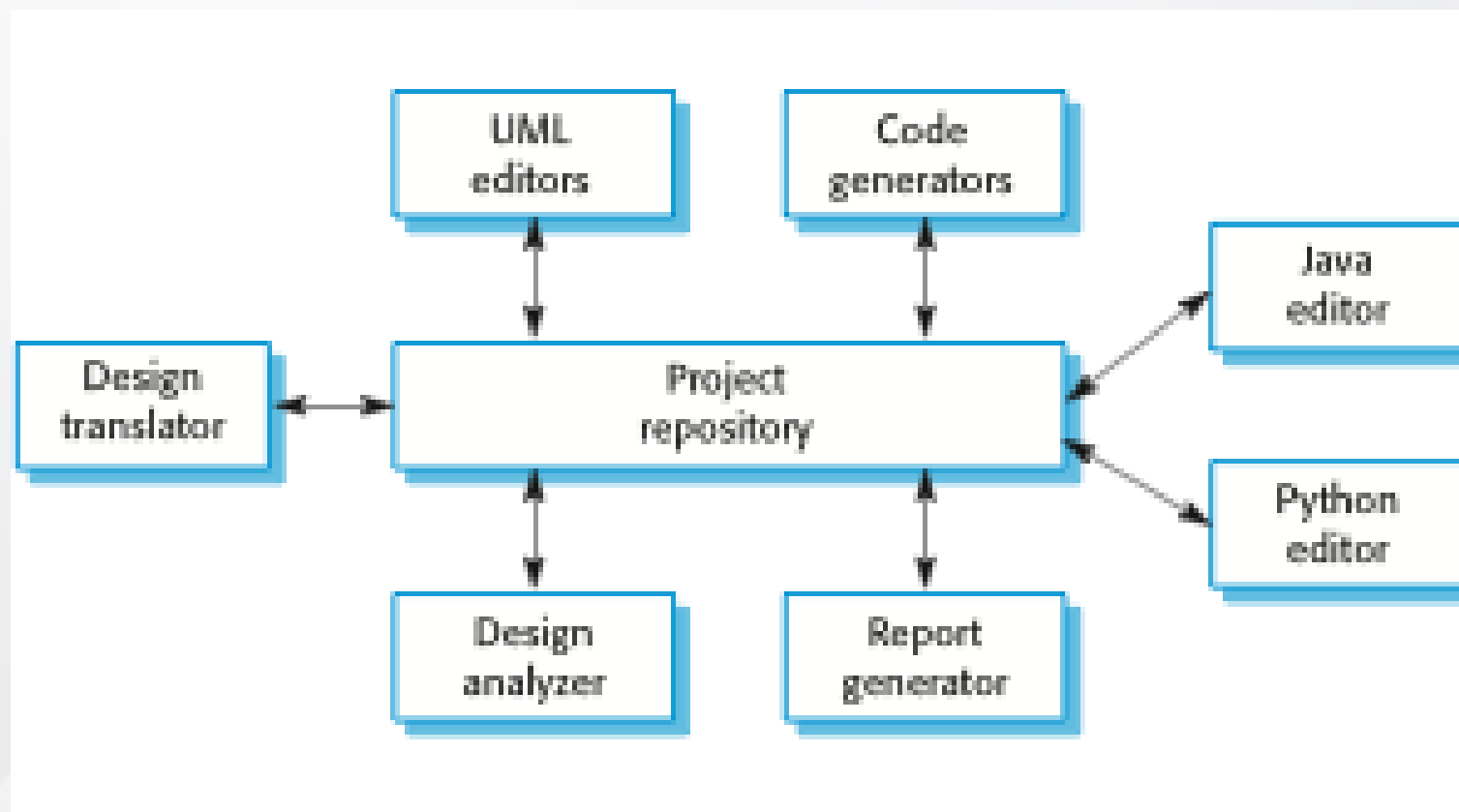
THE LAYERED ARCHITECTURE PATTERN

Lo. 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.
Example	A layered model of a system for sharing copyright documents held in different libraries.
When used	1. Used when building new facilities on top of existing systems; 2. When the development is spread across several teams with each team responsibility for a layer of functionality; 3. When there is a requirement for multi-level security.
Advantages	1. Allows replacement of entire layers as long as the interface is maintained. 2. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	1. In practice, providing a clean separation between layers is often difficult. 2. A high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. 3. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

REPOSITORY ARCHITECTURE

- Sub-systems must exchange data. This may be done in two ways:
 - Shared data is held in a central database or repository and may be accessed by all sub-systems;
 - 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 is most commonly used. This is an efficient data sharing mechanism.

A REPOSITORY ARCHITECTURE FOR AN IDE



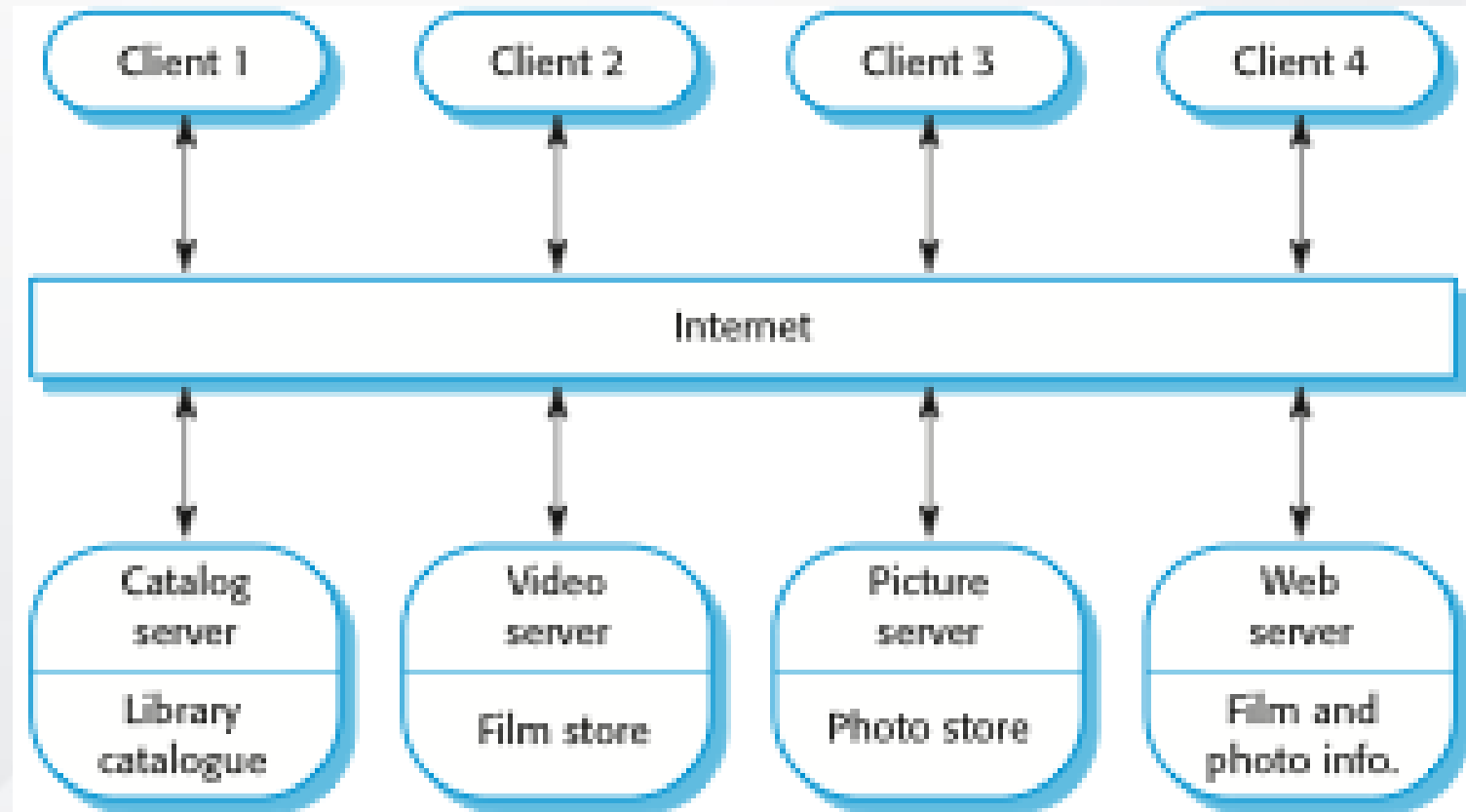
THE REPOSITORY PATTERN

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	An integrated development environment (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	1. when you have a system in which large volumes of information are generated that has to be stored for a long time. 2. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	1. Components can be independent—they do not need to know of the existence of other components. 2. Changes made by one component can be propagated to all components. 3. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	1. The repository is a single point of failure so problems in the repository affect the whole system. 2. May be inefficiencies in organizing all communication through the repository. 3. Distributing the repository across several computers may be difficult.

CLIENT-SERVER ARCHITECTURE

- Distributed system model which shows how data and processing is distributed across a range of components.
- Set of stand-alone servers which provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

A CLIENT-SERVER ARCHITECTURE FOR A FILM LIBRARY



THE CLIENT-SERVER PATTERN

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	A film and video/DVD library organized as a client–server system.
When used	1. Used when data in a shared database has to be accessed from a range of locations. 2. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	1. servers can be distributed across a network. 2. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	1. Each service is a single point of failure so susceptible to denial of service attacks or server failure. 2. Performance may be unpredictable because it depends on the network as well as the system. 3. May be management problems if servers are owned by different organizations.

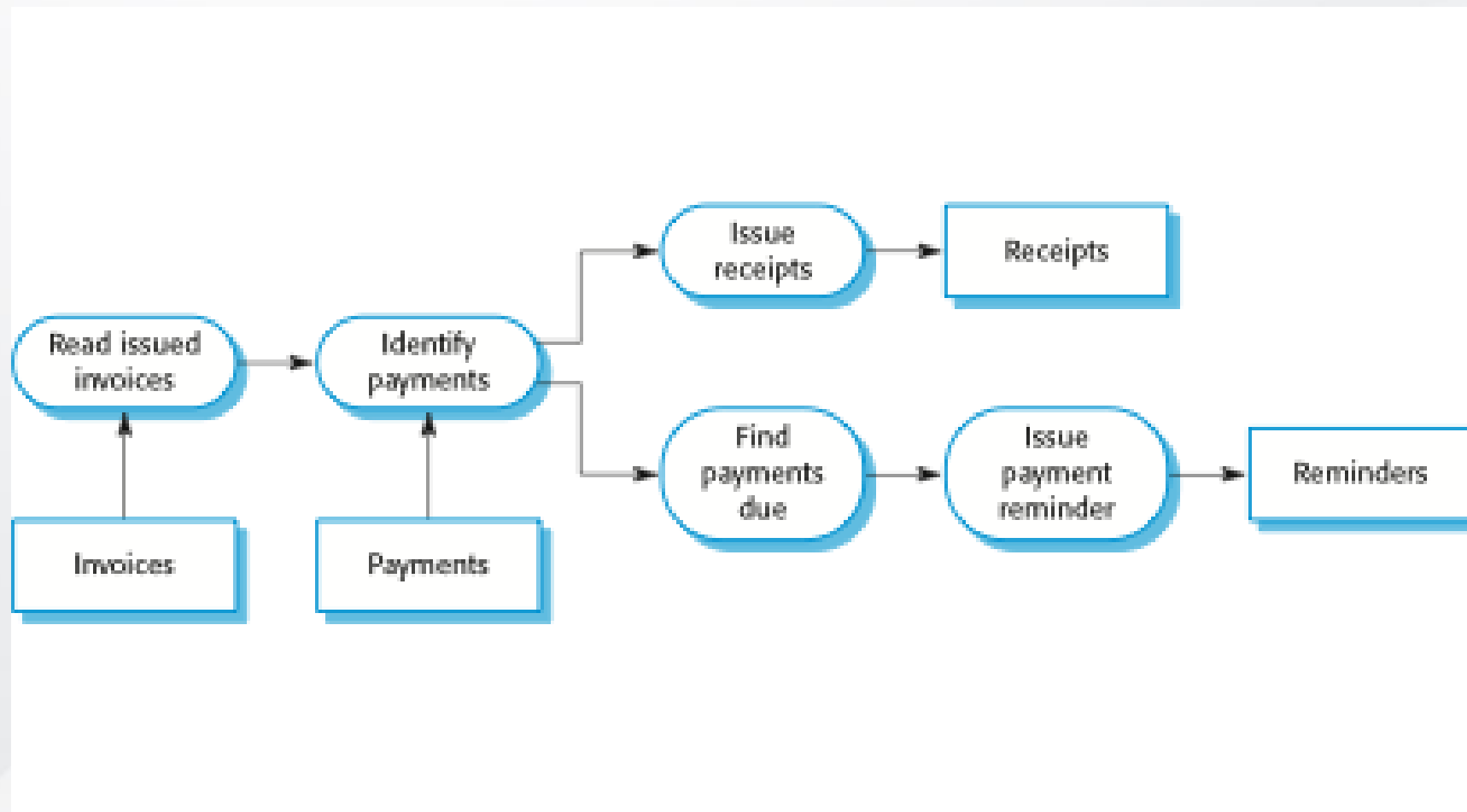
PIPE AND FILTER ARCHITECTURE

- 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.

THE PIPE AND FILTER PATTERN

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	processing invoices example .
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	1. Easy to understand and supports transformation reuse. 2. Workflow style matches the structure of many business processes. 3. Evolution by adding transformations is straightforward. 4. Can be implemented as either a sequential or concurrent system.
Disadvantages	1. The format for data transfer has to be agreed upon between communicating transformations. 2. Each transformation must parse its input and un-parse 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.

AN EXAMPLE OF THE PIPE AND FILTER ARCHITECTURE



APPLICATION ARCHITECTURES

APPLICATION ARCHITECTURES

- Application systems are designed to meet an organizational need.
- As businesses have much in common →
 - Their application systems also tend to have a common architecture.
- A generic application architecture
 - is an architecture that may be configured and adapted to create a system that meets specific requirements.

WHEN TO USE APPLICATION ARCHITECTURES ?

- As a starting point for architectural design.
- As a design checklist.
- As a way of organizing the teamwork
- As a means of assessing components for reuse.

EXAMPLES OF APPLICATION TYPES

Transaction processing applications

- Data-centered applications that ***process user requests*** and update information in a system database.

Language processing systems

- Applications where the ***users' intentions are specified in a formal language*** that is processed and interpreted by the system.

Data processing applications

- Data driven applications that process data in batches ***without explicit user intervention*** during the processing.

Event processing systems

- Applications where ***system actions depend on interpreting events*** from the system's environment.

APPLICATION TYPE EXAMPLES:

Transaction processing systems

- Information systems
- E-commerce systems;
- Reservation systems.

Language processing systems

- Compilers;
- Command interpreters.



SUMMARY

- Architectural Design
- Software Architecture
- Application Architecture



Thank You

