Software Engineering

Chapter 6 – Architectural Design

Different Types of Software Architecture

Topics covered

- The requirements describe the function of a system as seen by the client.
- Given a set of requirements, the software development team must design a system that will meet those requirements.
- In this chapter, we look at the following aspects of design:
  - Software Architecture Definition
  - Architectural design decisions
  - Architectural Styles

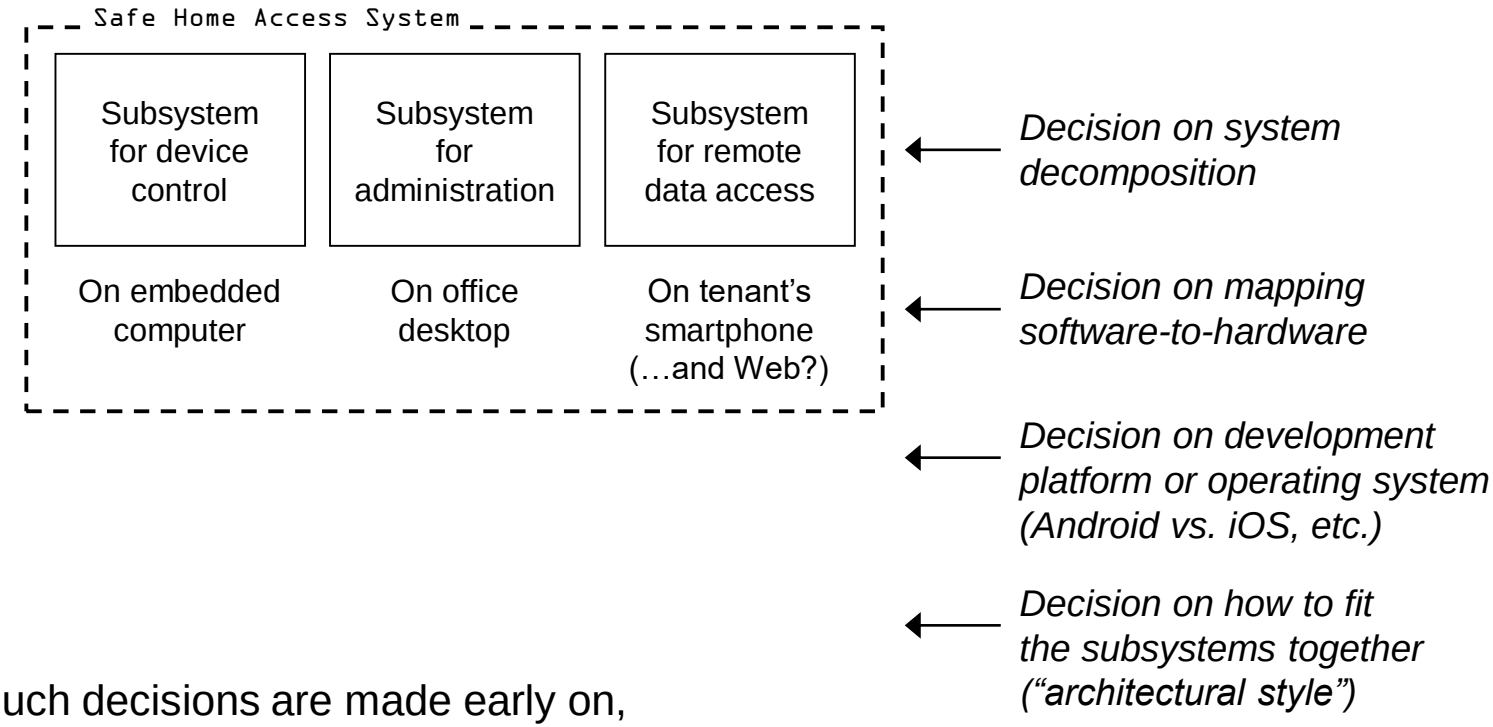
Software Architecture Definition

✧ **Software Architecture:** fundamental structures of a software system and the discipline of building such structures and systems.

- Software architecture, describing the subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, and data storage
- Software architecture is a set of high-level **decisions** that determine the structure of the solution (parts of system-to-be and their relationships)
- Decisions to use well-known solutions that are proven to work for similar problems

Example Architectural Decisions

Example decisions:



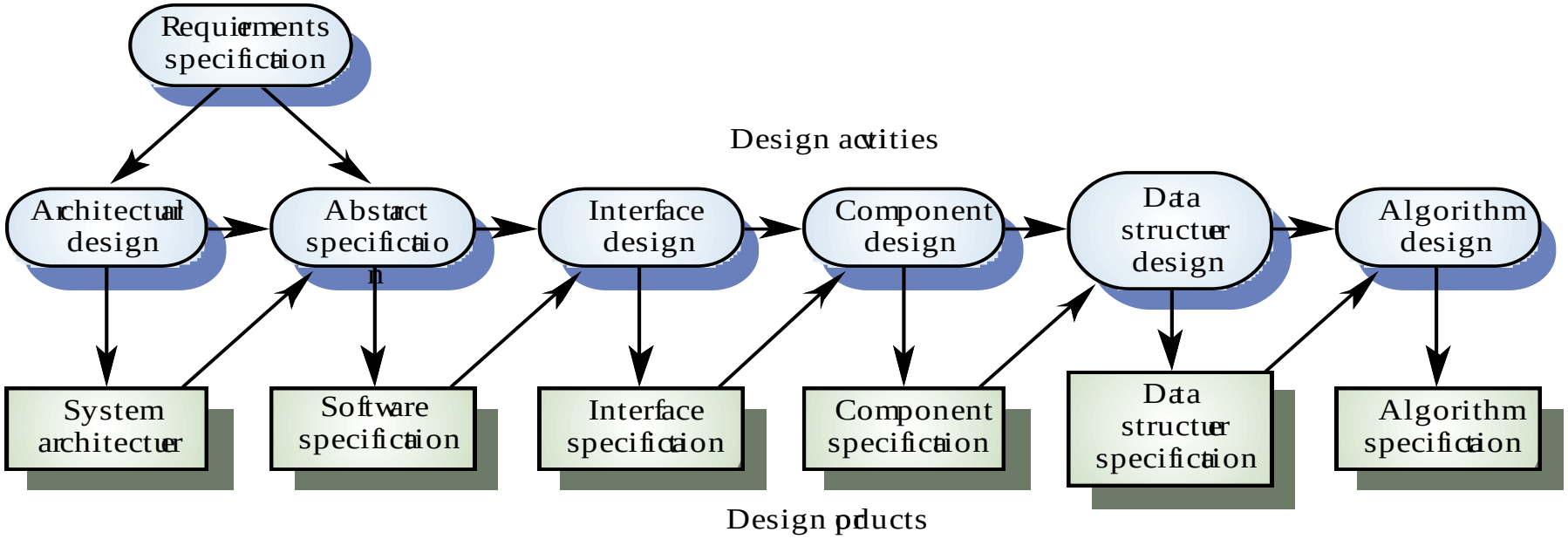
Such decisions are made early on,

perhaps while discussing the requirements with the customer

to decide which hardware devices will be used for user interaction and device control

The Design Process

- The **system** should be described at several different levels of abstraction.
- **Design takes place in overlapping stages.**



Architectural design

- Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system.
- Architectural design is the critical link between design and requirements engineering, as it identifies the main structural components in a system and the relationships between them.
- Architecture design plans for how the system will be distributed across computers and what hardware and software will be used for each computer.
- The output of the architectural design process is an architectural model that describes how the system is organized as a set of communicating components.

Architecture versus Design

- **Architecture** focuses on non-functional requirements ("cross-cutting concerns") and decomposition of functional requirements
- **Design** focuses on implementing the functional requirements
- **Software Design**: Deriving a solution which satisfies software requirements

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.
- 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.

Architectural Components

- Software systems can be divided into four basic functions:
  1. Data storage.
  2. Data access logic: the processing required to access stored data.
  3. Application logic: the logic documented in the use cases, and functional requirements.
  4. Presentation logic: the display of information to the user and the acceptance of the user's commands.

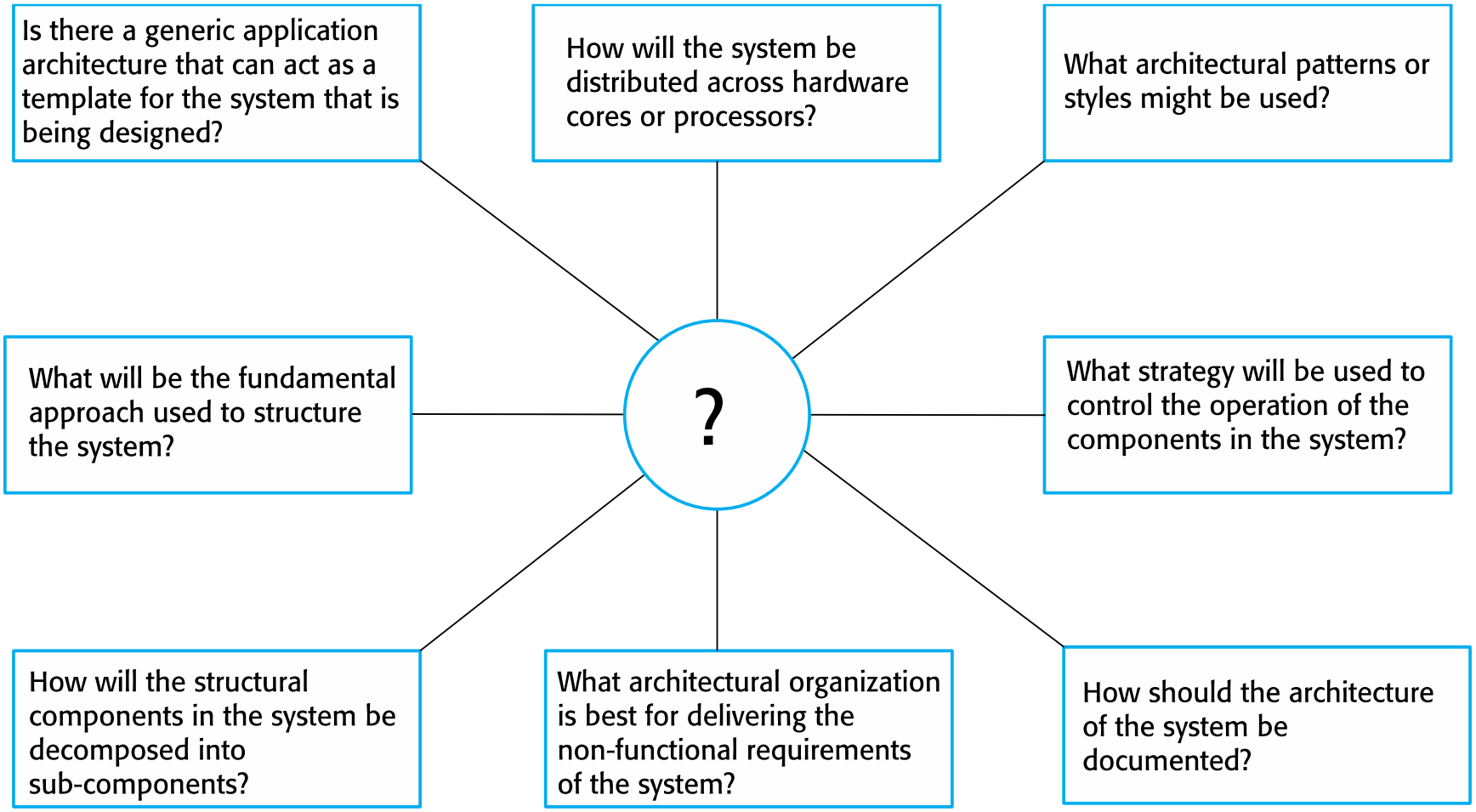
Architectural Components (cont'd)

- The three primary hardware components:
  - Client computers: Input-output devices employed by users (e.g., PCs, laptops, handheld and mobile devices, smart phones)
  - Servers: Larger multi-user computers used to store software and data.
  - Network: Connects the computers.

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.

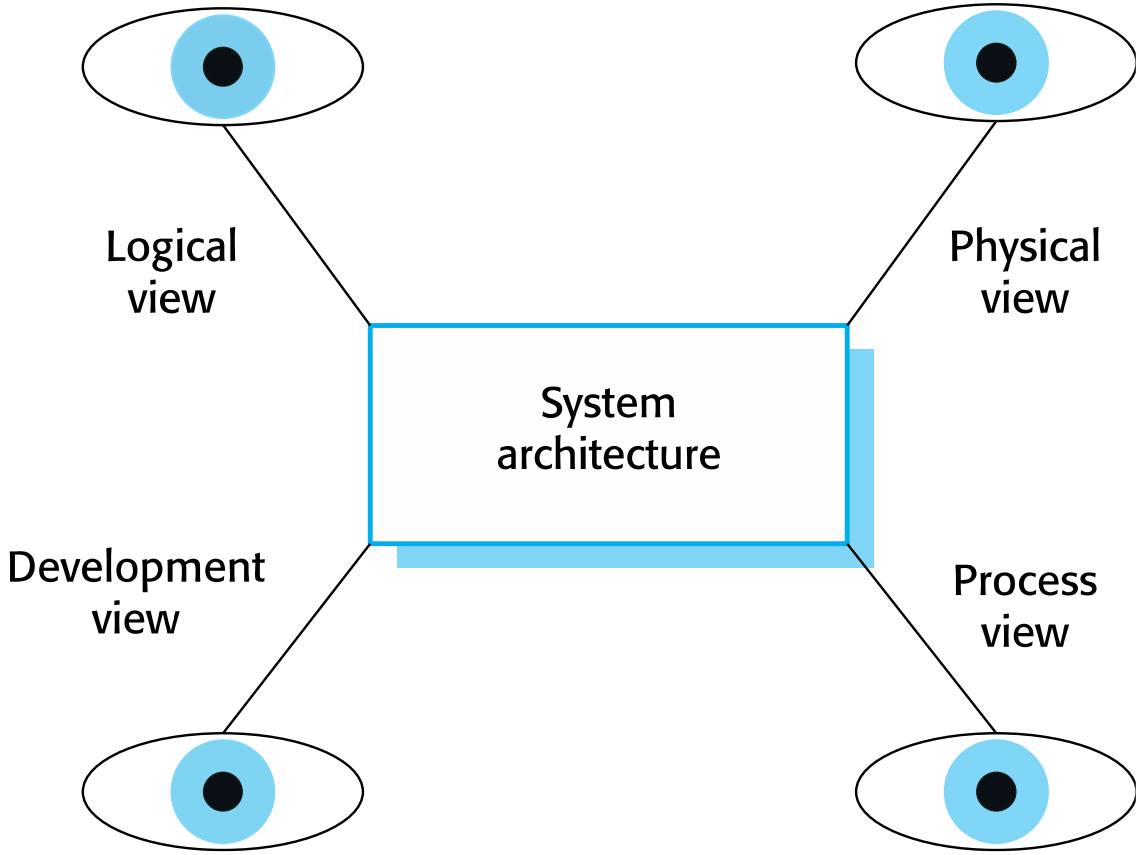
Architectural design decisions



Architectural views

- What views or perspectives are useful when designing and documenting a system's architecture?
- What notations should be used for describing architectural models?
- Views are different kinds of "blueprints" created for the system-to-be
  - E.g., blueprints for buildings: construction, plumbing, electric wiring , heating, air conditioning, …
    (Different stakeholders have different information needs)
- It is impossible to represent all relevant information about a system's architecture in a single diagram, as a graphical model 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.

Architectural views



4 + 1 view model of software architecture

- A logical view, which shows the key abstractions in the system as objects or object classes. It should be possible to relate the system requirements to entities in this logical view.
- A process view, which shows how, at run-time, the system is composed of interacting processes. This view is useful for making judgments about non-functional system characteristics such as performance and availability.
- A development view, which shows how the software is decomposed for development; that is, it shows the breakdown of the software into components that are implemented by a single developer or development team.
- 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)

How to Fit Subsystems Together: Some Well-Known Architectural Styles

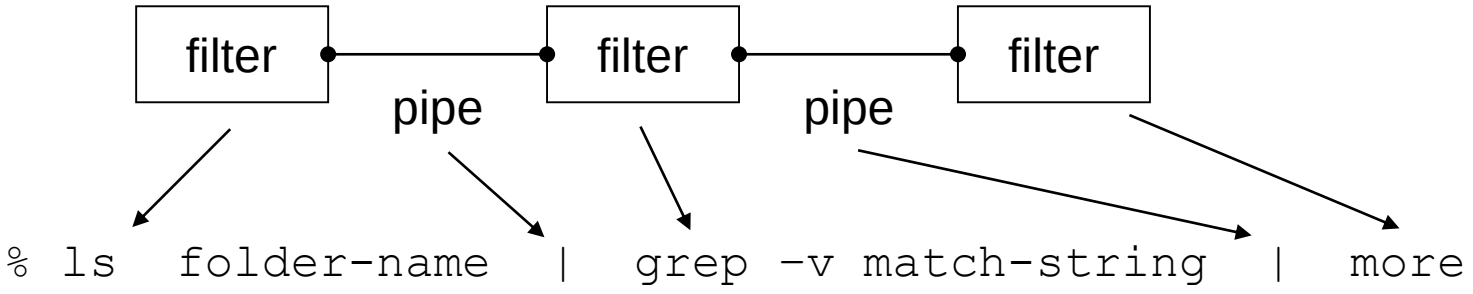
- UNIX shell script architectural style: Pipe-and-Filter
- Client/Server
- Central Repository (database)
- Layered (or Multi-Tiered)
- Model-View-Controller
- World Wide Web architectural style: REST (Representational State Transfer)

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.
- In this section, we introduce Architectural patterns and briefly describe a selection of Architectural patterns that are commonly used.

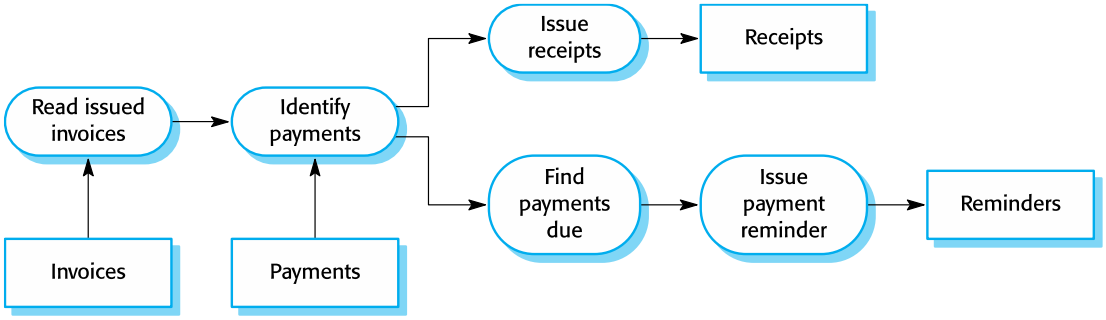
Pipe and filter architecture

- Functional transformations process their inputs to produce outputs. Thus, output from one subsystem is the input to the next.
- May be referred to as a pipe and filter model (as in UNIX shell).
- 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.
  - Components: **Filters** transform input into output
  - Connectors: **Pipe** data streams
  - Example: UNIX shell commands



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	The Figure below is an example of a pipe and filter system used for processing invoices (payment/ billing system).
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. 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.

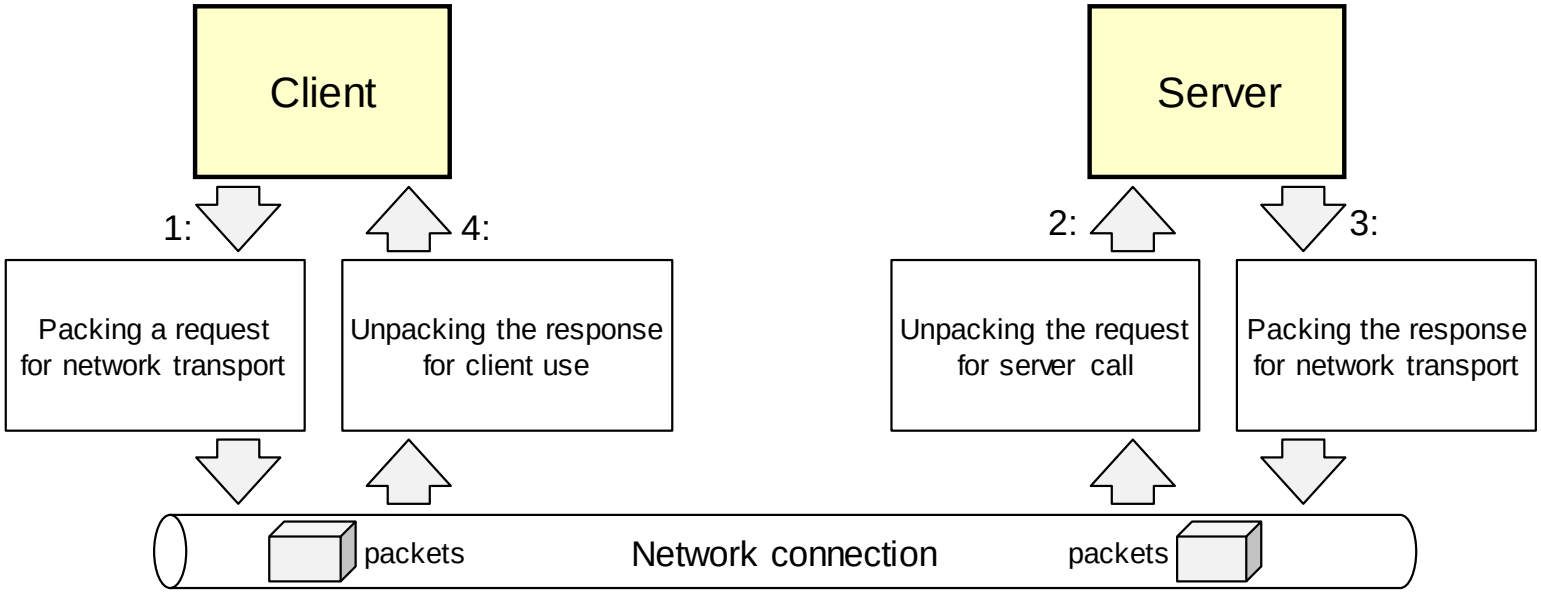


Client-server architecture

- Distributed system model which shows how data and processing is distributed across a range of components.
  - Can be implemented on a single computer.
- 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.
- Client-server architectures balance the processing between client devices and one or more server devices.
- Generally, clients are responsible for the presentation logic, and the server(s) are responsible for the data access logic and data storage.
- Application logic location varies depending on the C-S configuration chosen.

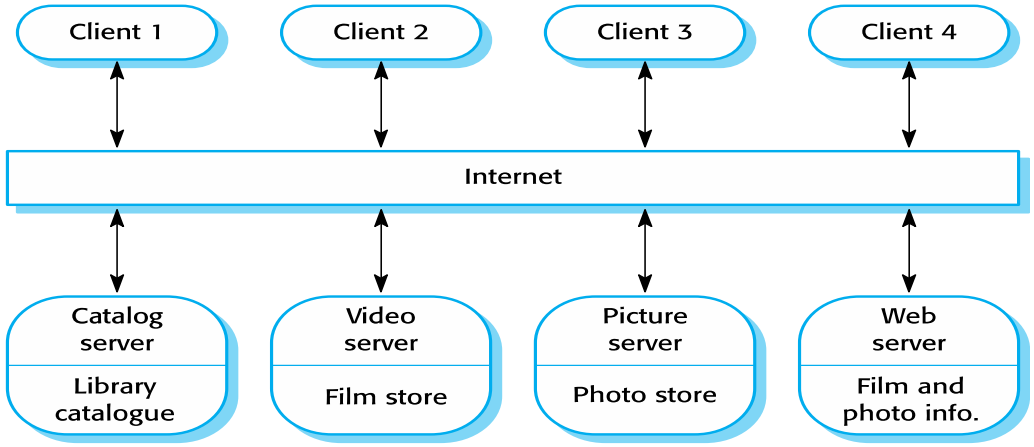
Architectural Style: Client/Server

- A **client** is a triggering process; a **server** is a reactive process. Clients make requests that trigger reactions from servers.
- A **server component**, offering a set of services, listens for requests for those services. It waits for requests and then reacts to them.
- A **client component**, desiring that a service be performed, sends a request to the server via a **connector**.
- The server either rejects or performs the request and sends a response back to the client.



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	The Figure below 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 (scalable). 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.



Two-Tiered Client-Server Architecture

- Thick client – most of application logic on the client side (shown here)
- Thin client – little application logic on the client side; most shifted to server side

Three-Tiered Client-Server Architecture

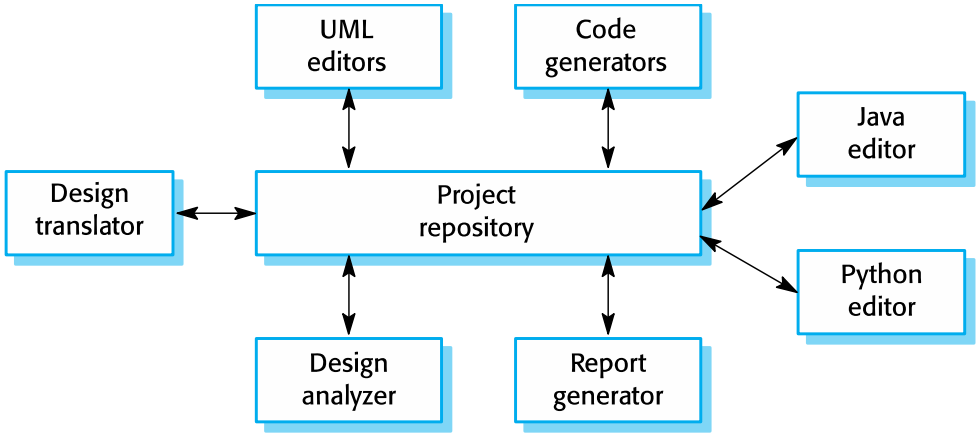
- Adds “specialized” servers – one for application logic; one for data base tasks

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 as this is an efficient data sharing mechanism.

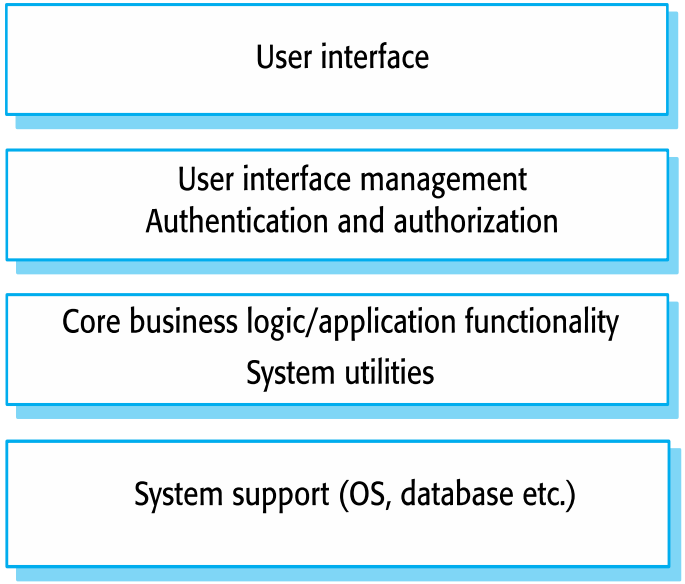
The Repository pattern

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	The Figure below 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.



Layered architecture

- ✧ Used to model the interfacing of sub-systems.
- ✧ Organises the system into a set of layers (or abstract machines) 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.



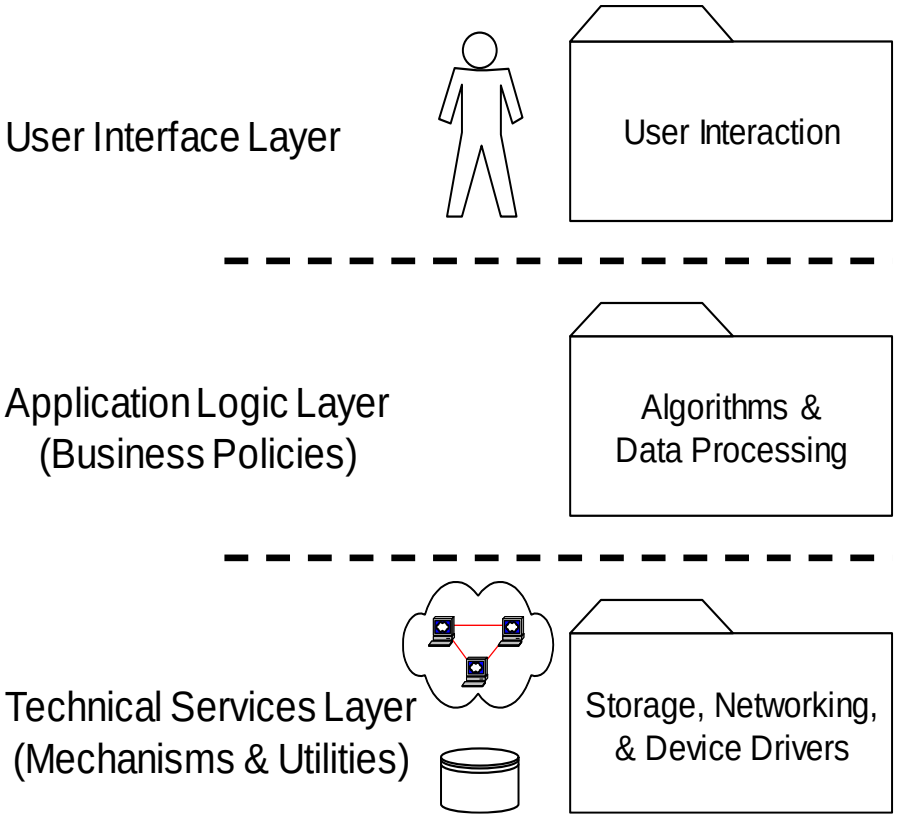
The Layered architecture pattern

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.

Architectural Style: Layered

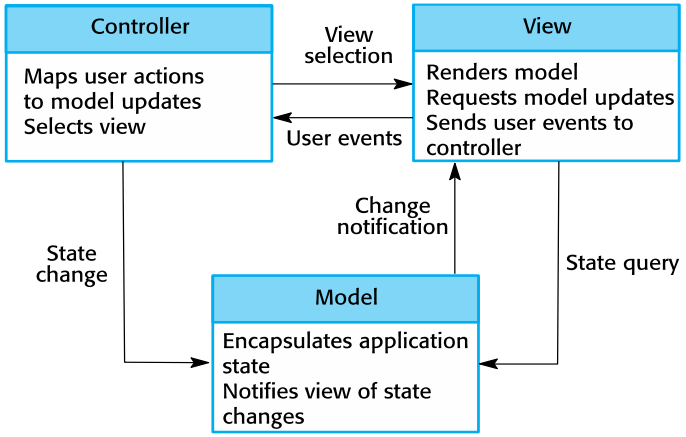
a.k.a. Tiered Software Architecture

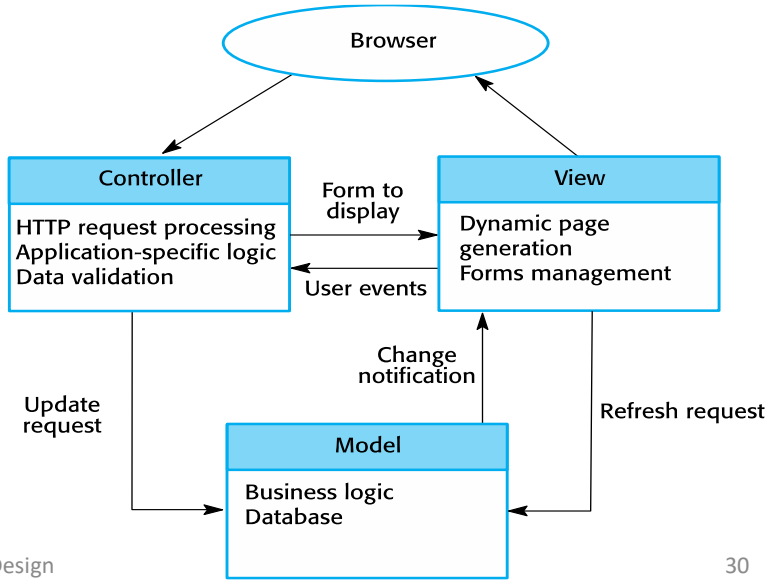
- A layered system is organized hierarchically, each layer providing services to the layer above it and using services of the layer below it



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.
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.





Architectural Style: Model-View-Controller

- **Model**: holds all the data, state and application logic. Unaware of the View and Controller.
- **View**: gives user a presentation of the Model. Gets data directly from the Model
- **Controller**: Takes user input and figures out what it means to the Model

REST: Representational state transfer

- REST (REpresentational State Transfer) is an architectural style for developing web services
- In the REST architectural style, the implementation of the client and the implementation of the server can be done independently without each knowing about the other. This means that the code on the client side can be changed at any time without affecting the operation of the server, and the code on the server side can be changed without affecting the operation of the client.
- REST is intended to evoke an image of how a well-designed Web application behaves:
  - a network of web pages (a virtual state-machine),
  - where the user progresses through an application by selecting links (state transitions),
  - resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.