Chapter 6 - Architectural Design

4+1架构视图
架构模式-情况
MVC记细节
四个设计原则

MI Qing (Lecturer)

Telephone: 15210503242

Email: miqing@bjut.edu.cn

Office: 410, Information Building

Topics Covered

- ✧ Architectural views
- ✧ Architectural patterns
- ✧ Architectural design decisions

Analysis VS Design

Analysis

Asks “what is the problem?”

- What happens in the current system?
- What is required in the new system?

Results in a detailed understanding of:

- Requirements
- Domain properties

Focuses on the way human activities are conducted

Design

Investigates “how to build a solution”

- How will the new system work?
- How can we solve the problem that the analysis identified?

Results in a solution to the problem:

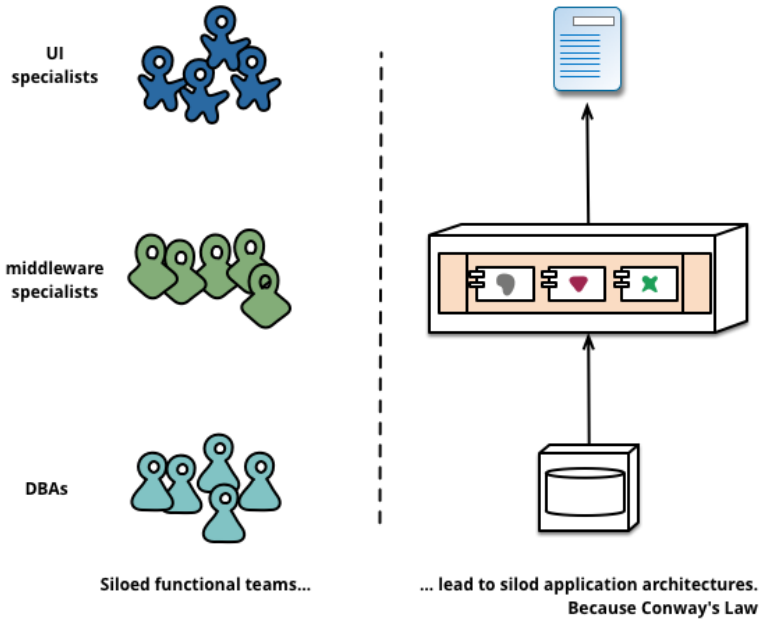
- A working system that satisfies the requirements
- Hardware, software, peopleware

Focuses on building technical solutions

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.

Architectural Design

- A software architecture defines:
  - The components of the software system
  - How the components use each other's functionality and data
  - How control is managed between the components
- Conway's Law:
  - The structure of a software system reflects the structure of the organization that built it.
- The output is an architectural model that describes how the system is organized as a set of communicating components.

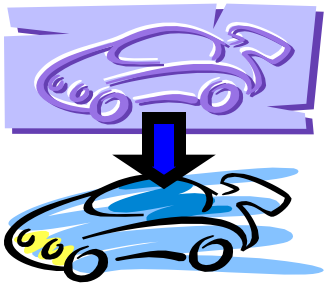


Four Design Philosophies

Decomposition & Synthesis

- Drivers
  - Managing complexity
  - Reuse
- Example
  - Design a car by designing separately the chassis, engine, drivetrain, etc. Use existing ***components*** where possible

Search



- Drivers
  - Transformation
  - Heuristic evaluation
- Example
  - Design a car by ***transforming*** an initial rough design to get closer and closer to what is desired

Four Design Philosophies

Negotiation

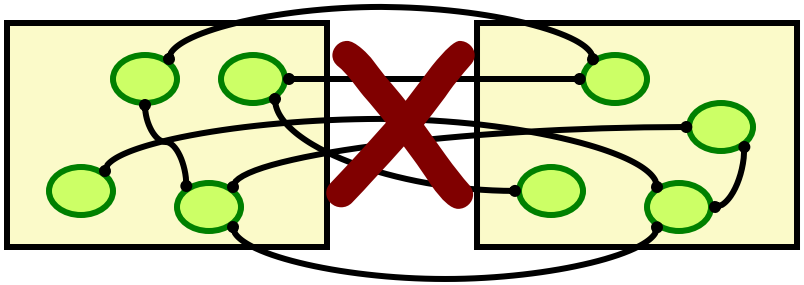
- Drivers
  - Stakeholder conflicts
  - Dialogue process
- Example
  - Design a car by getting *each stakeholder* to suggest (partial) designs, and then compare and discuss them

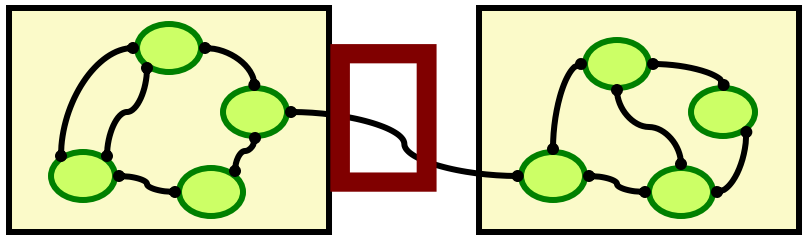
Situated Design

- Drivers
  - Errors in existing designs
  - Evolutionary change
- Example
  - Design a car by observing what's wrong with existing cars *as they are used*, and identifying improvements

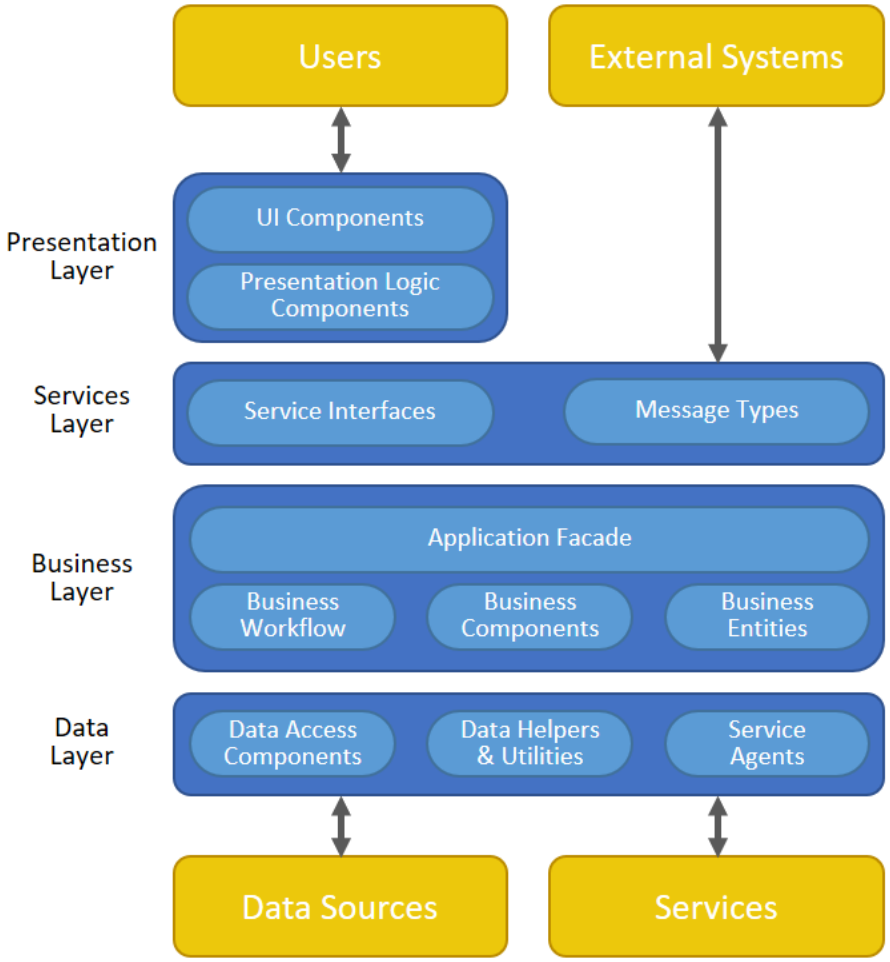
Coupling and Cohesion

- A good architecture minimizes coupling between modules:
  - Goal: modules don't need to know much about one another to interact
  - Low coupling makes future change easier
- A good architecture maximizes the cohesion of each module:
  - Goal: the contents of each module are strongly inter-related
  - High cohesion makes a module easier to understand

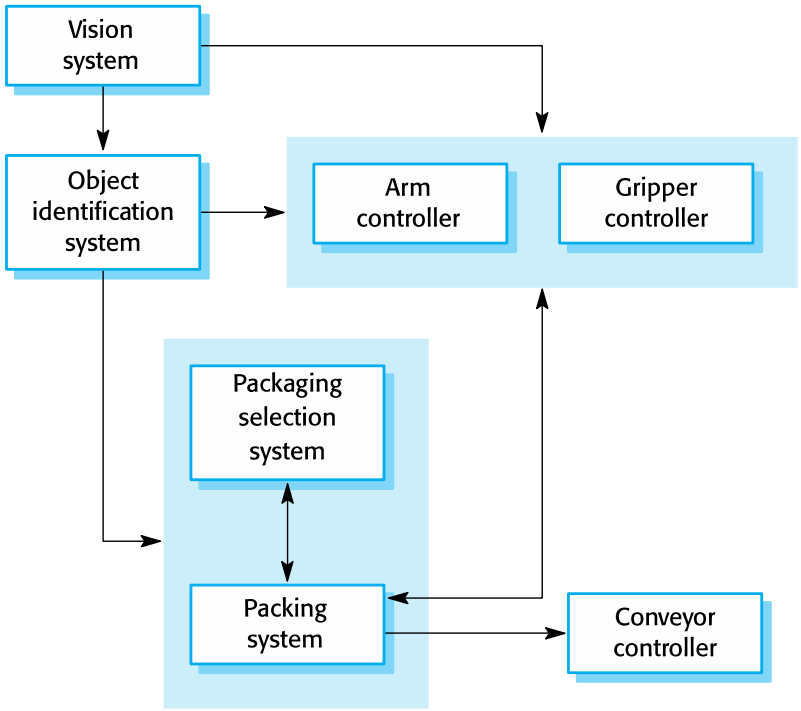




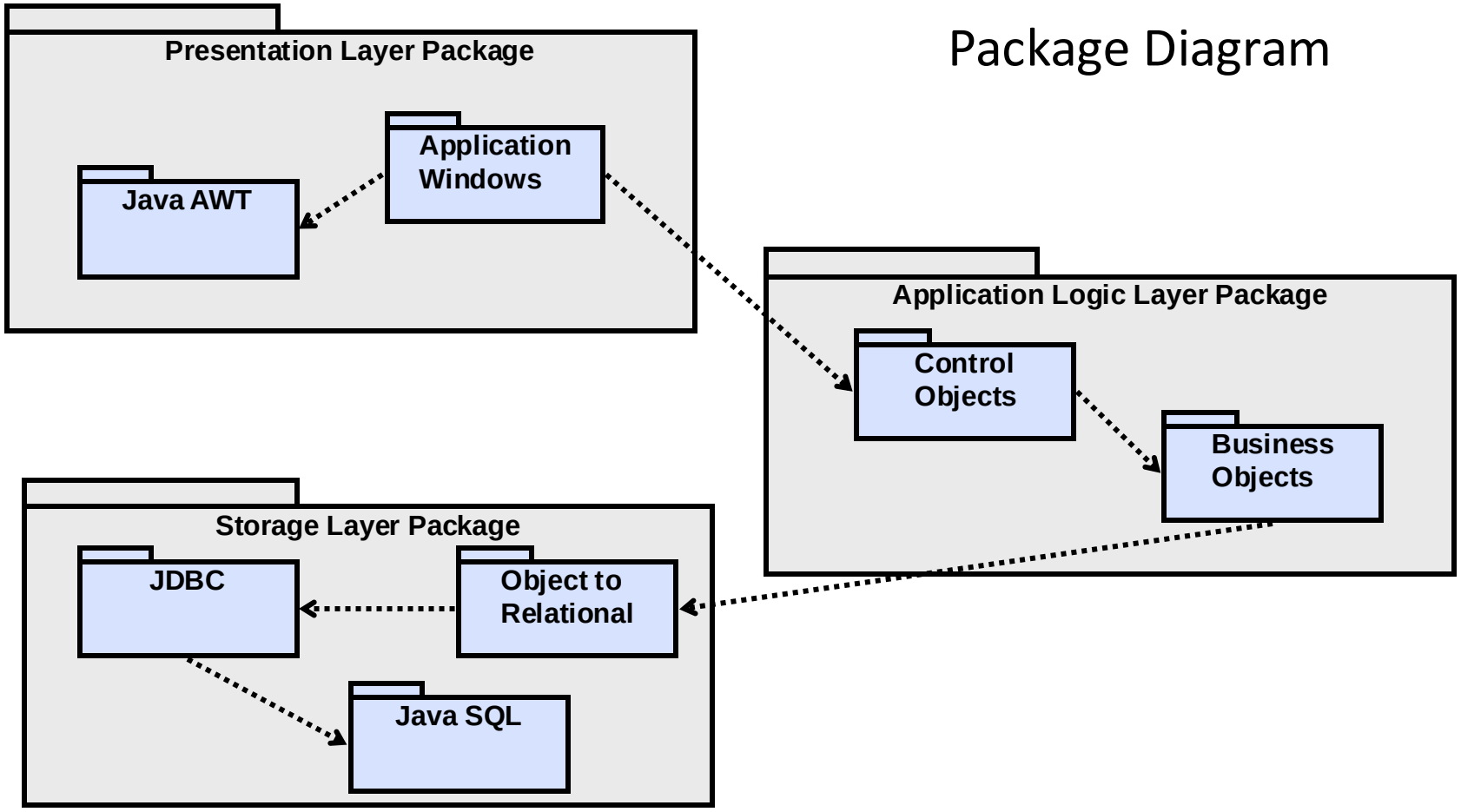
Architectural Representations



Box-and-line Diagram



Architectural Representations



Advantages of Explicit Architecture

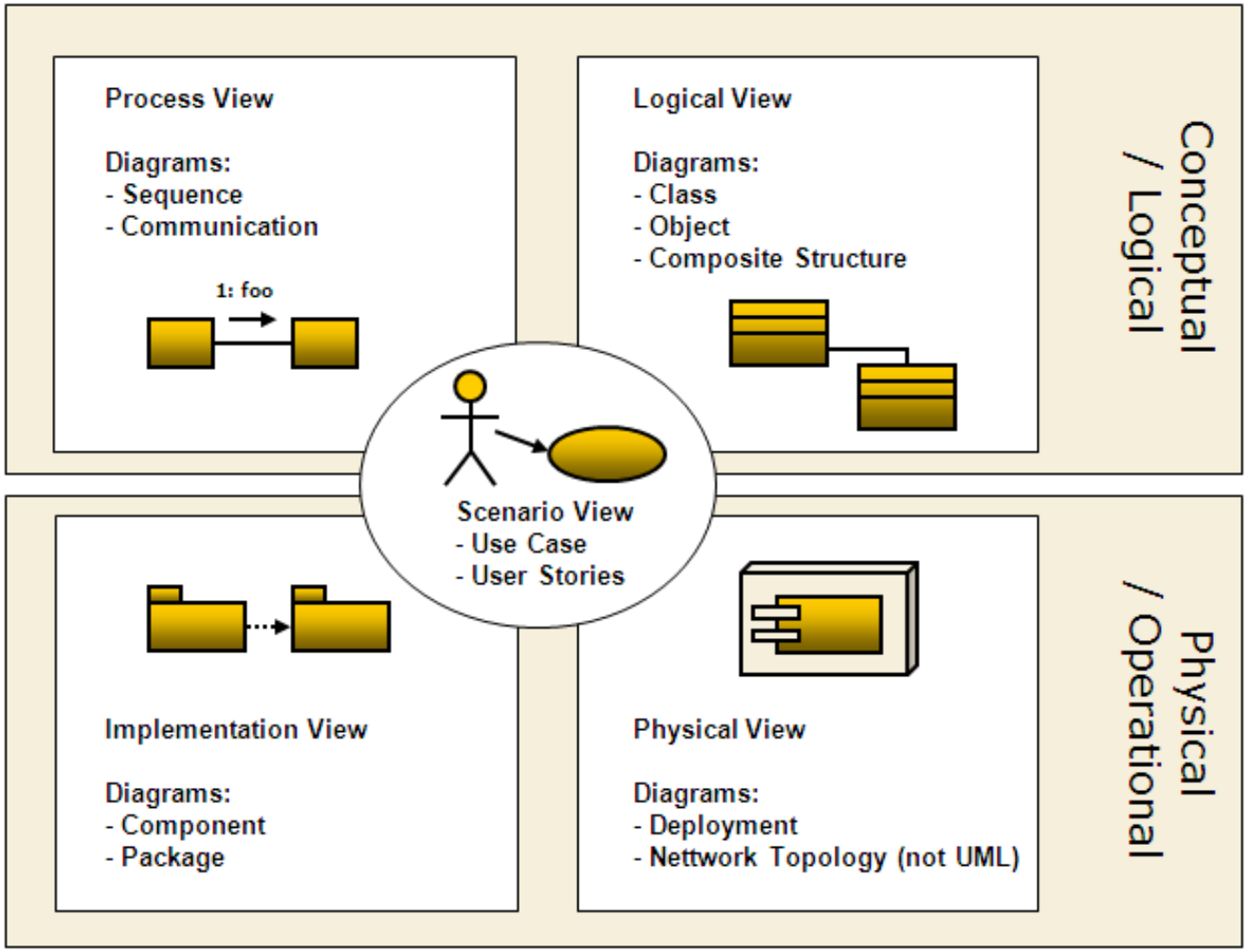
- ✧ Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- ✧ System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- ✧ Large-scale reuse
  - The system architecture is often the same for systems with similar requirements and so can support large-scale software reuse.

Architectural Views

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?
- 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.

4+1 View Model of 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.

Architectural Patterns

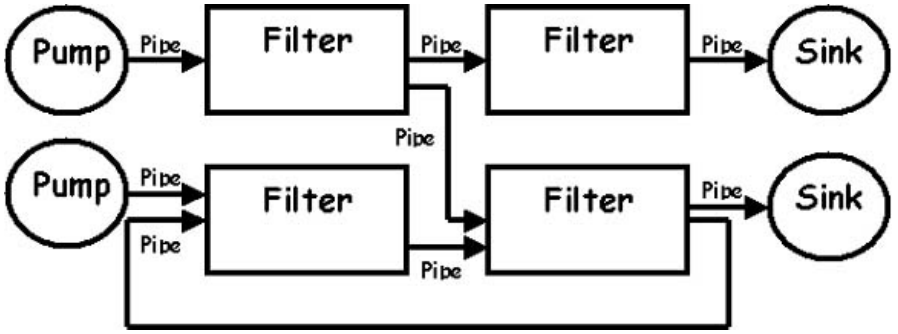
给情况写对应架构模式

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.

Pipe-and-filter Architecture

- The filter transforms or filters the data it receives via the pipes with which it is connected.
- The pipe is the connector that passes data from one filter to the next.
  - It is a directional stream of data, that is usually implemented by a data buffer to store all data, until the next filter has time to process it.
- The pump or producer is the data source.
  - It can be a static text file, or a keyboard input device, continuously creating new data.
- The sink or consumer is the data target.
  - It can be a another file, a database, or a computer screen.



Pipe-and-filter Architecture

Not really suitable for interactive systems.

✧ Examples:

- Unix programs. The output of one program can be linked to the input of another program.
- Compilers. The consecutive filters perform lexical analysis, parsing, semantic analysis, and code generation.

✧ Interesting properties:

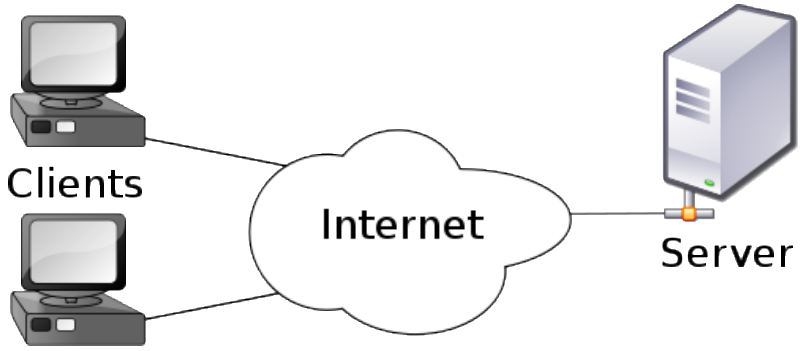
- Filters don't need to know anything about what they are connected to
- Filters can be implemented in parallel
- Specialized analysis such as throughput and deadlock analysis is possible

Pipe-and-filter Architecture

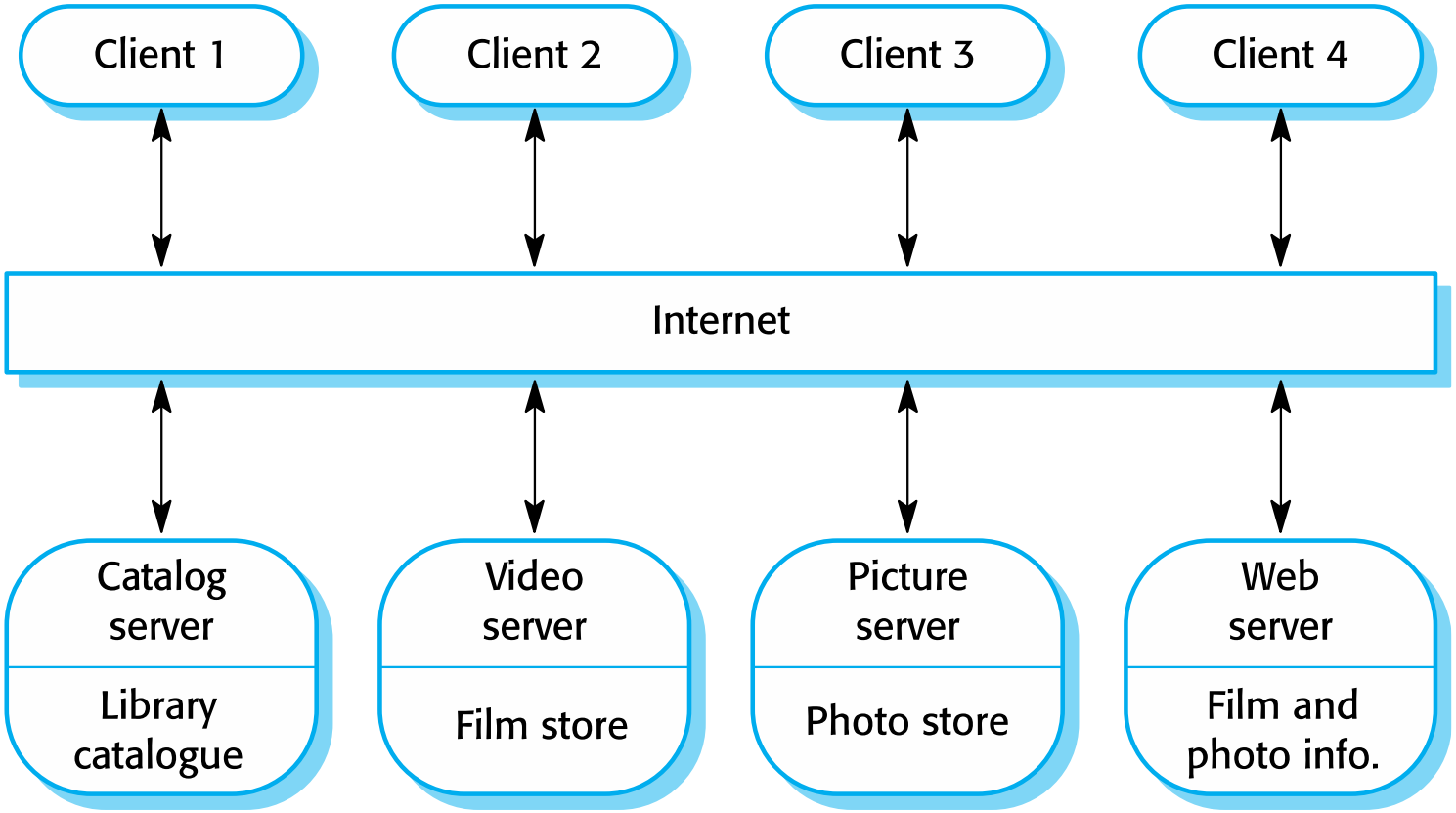
Name	Pipe-and-filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
When used	Commonly used in data processing 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.

Client-server Architecture

- A system that follows the client-server pattern is organized as a set of services and associated servers, and clients that access and use the services.
- The major components of this model are:
  - A set of stand-alone servers which provide specific services such as printing, data management, etc.
  - A set of clients which call on these services.
  - A network which allows clients to access servers.



An Example of Client-server Architecture

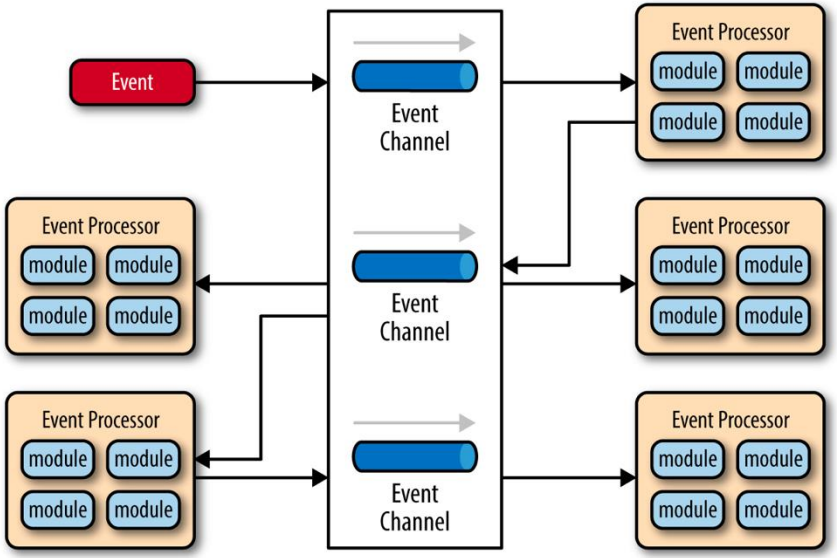


Client-server Architecture

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.
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.

Event-based Architecture

- ✧ Examples
  - Debugging systems (listen for particular breakpoints)
  - Graphical user interfaces
- ✧ Interesting properties
  - Announcers of events don't need to know who will handle the event
  - Support re-use, and evolution of systems
- ✧ Disadvantages
  - Components have no control over ordering of computations

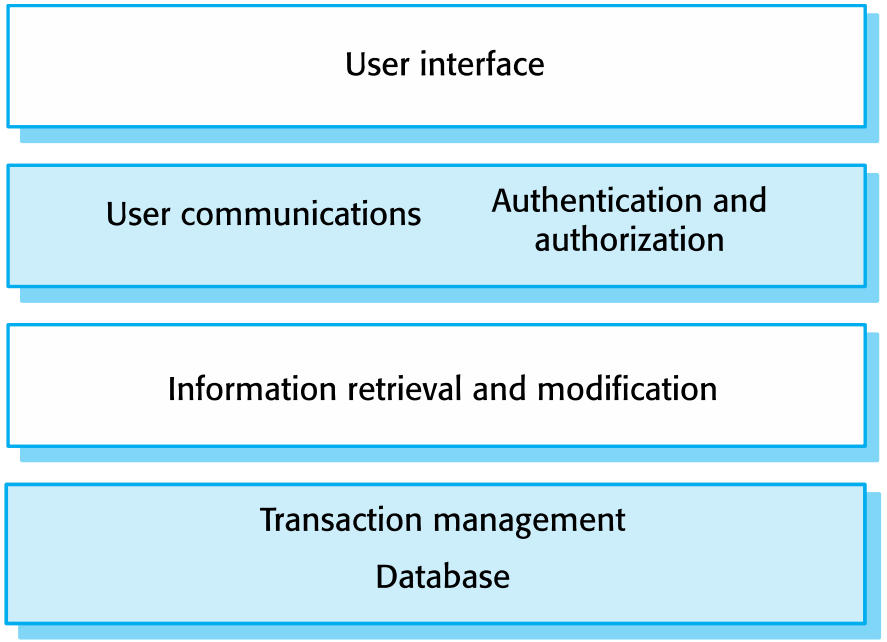


Event-based Architecture

- There are several types of communication that may occur on the event bus:
  - Publish-Subscribe: Modules may subscribe to certain message types. Whenever a module publishes a message to the bus, it will be delivered to all modules that subscribed to its message type.
  - Broadcast: The message will be delivered to all (other) modules.
  - Point-to-point: The message has one and only one recipient
- Use it when your application can be factored in functionally separable modules that are capable of communicating through simple messages.

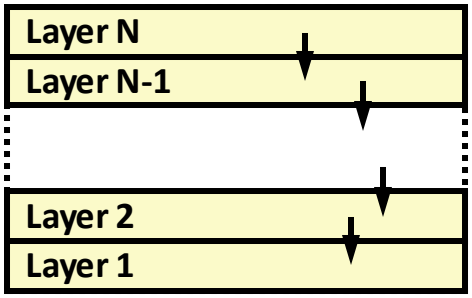
Layered Architecture

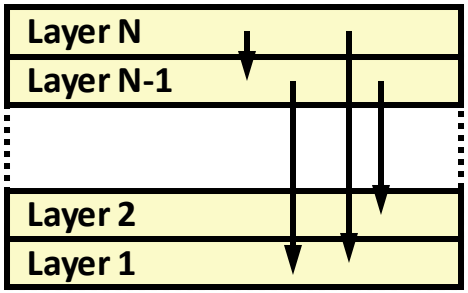
- Organises the system into a set of layers each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers.
- When a layer interface changes, only the adjacent layer is affected.



Layered Architecture

- ✧ Examples
  - Operating systems
  - Communication protocols





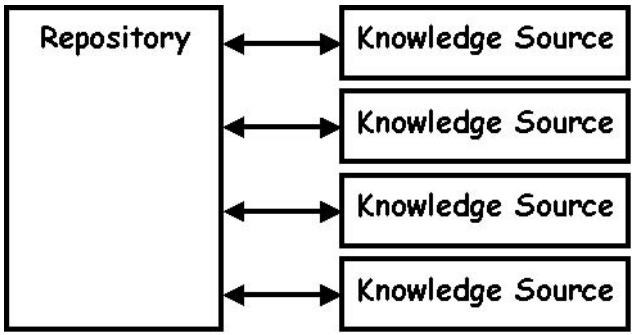
- ✧ Interesting properties
  - Support increasing levels of abstraction during design
  - Support enhancement (add functionality) and re-use
  - Can define standard layer interfaces
- ✧ Disadvantages
  - May not be able to identify (clean) layers

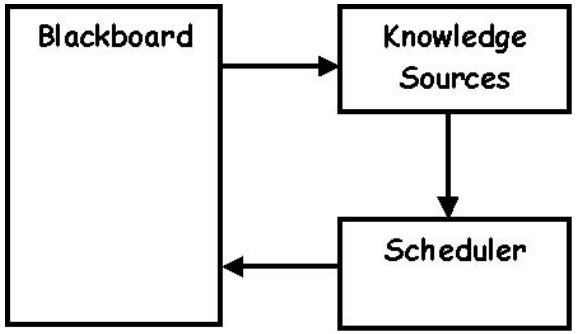
Layered Architecture

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.
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.

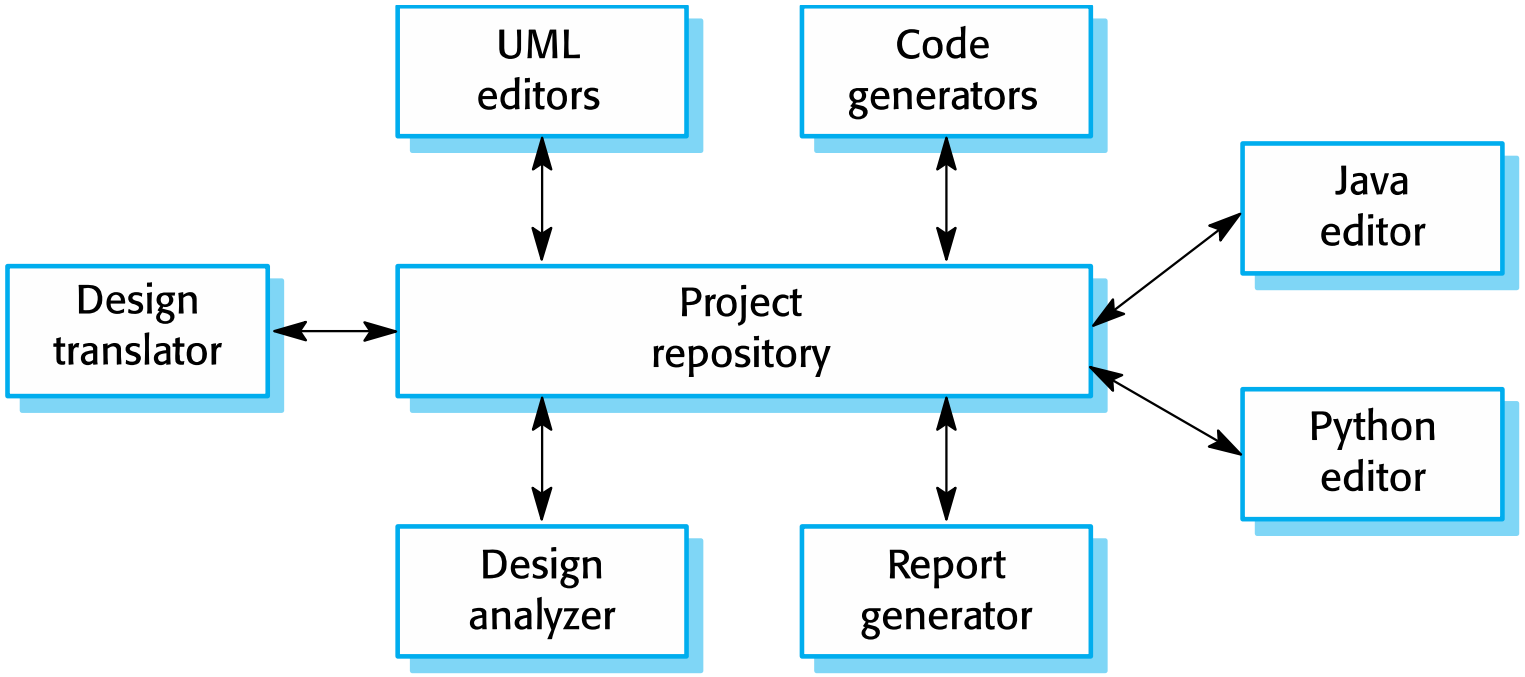
Repository Architecture

- The repository contains a single datastructure called Repository and a number of modules called Knowledge Sources that modify this datastructure.
- Examples
  - Database
  - Programming environments
- Disadvantages
  - Requires special attention if the repository is approached in parallel by several knowledge sources.





An Example of Repository Architecture

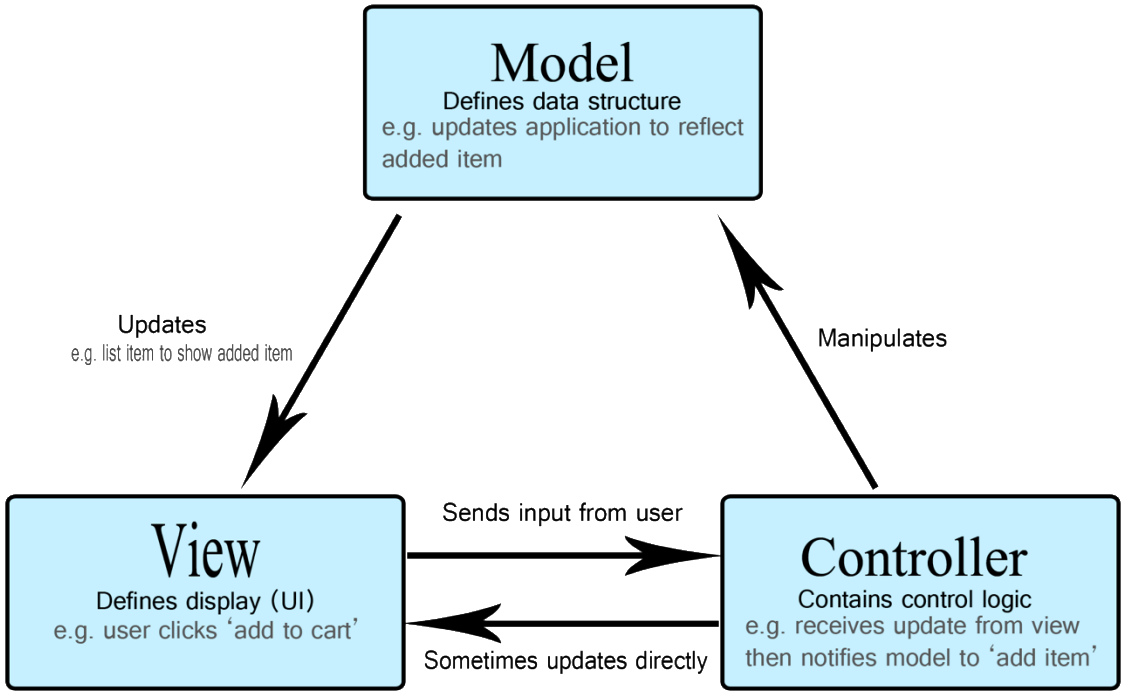


Repository Architecture

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.
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.

Model-View-Controller Architecture

- This architecture is used in simple GUI applications.
- The architecture is event-driven, which means that all activity starts by some event and is propagated by some other events.



MVC要记细节

Model-View-Controller Architecture

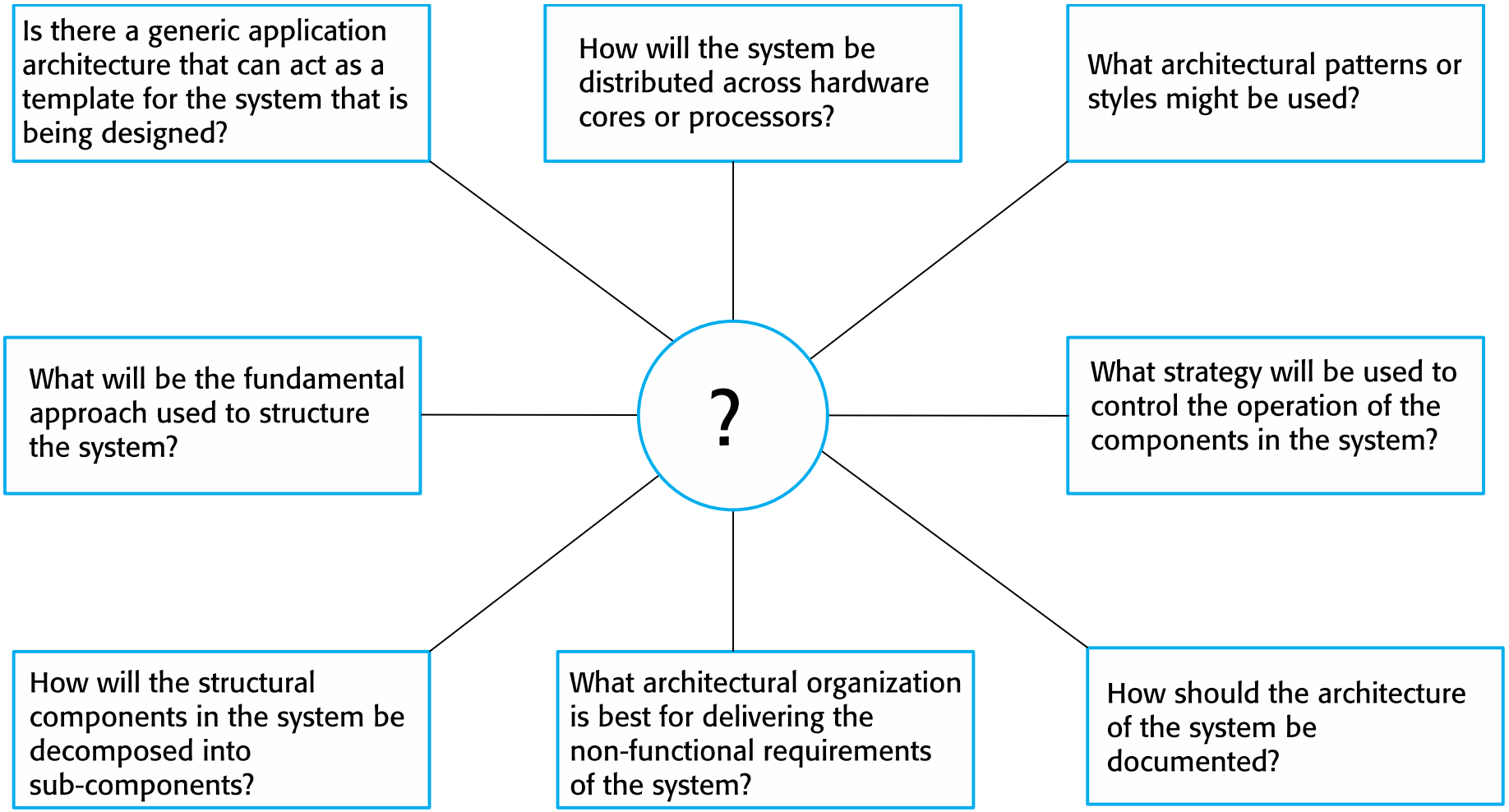
Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines 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 Design Decisions

Architectural Design Decisions

- Architectural design is a creative process in which you design a system organization that will satisfy the functional and non-functional requirements of a system.
  - There is no formulaic architectural design process.
- It is best to consider architectural design as a series of decisions to be made rather than a sequence of activities.
- Systems in the same domain often have similar architectures that reflect domain concepts.
  - The architecture of a system may be designed around one of more architectural patterns or styles.

Architectural Design Decisions



Architectural Design Decisions

四个设计原则要背

- Performance
  - Localize critical operations and minimize communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Availability
  - Include redundant components and mechanisms for fault tolerance.
- Maintainability
  - Use fine-grain, replaceable components.

Summary

Key Points

- A software architecture is a description of how a software system is organized.
- Architectures may be documented from several different perspectives or views such as a conceptual view, a logical view, a process view, and a development view.
- Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used and describe its advantages and disadvantages.
- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used.