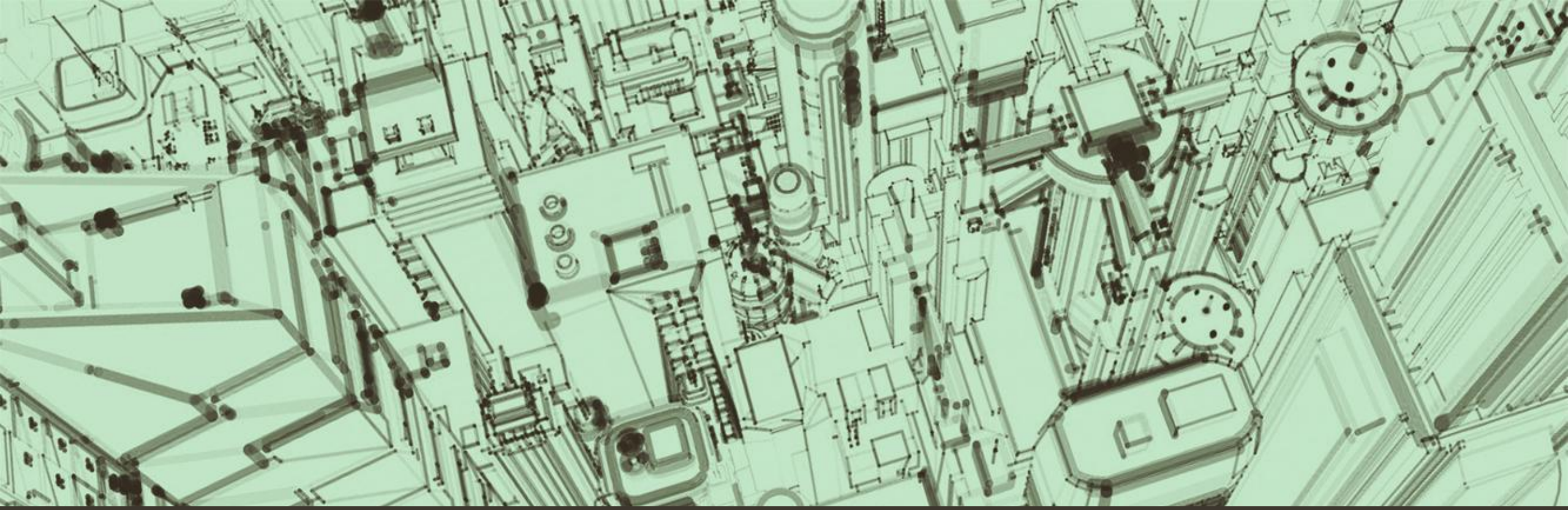


Software Engineering

Sobia Iftikhar
Sobia.Iftikhar@nu.edu.pk

Week 08

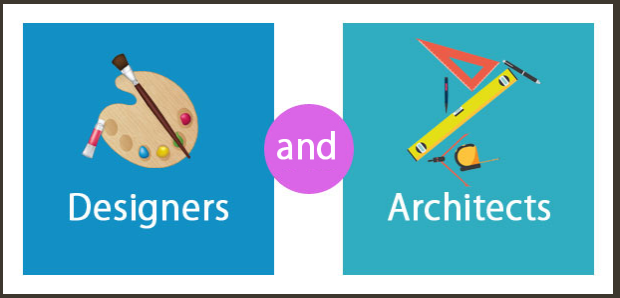
Class 19
31-March-2021

Content

- Architectural design decisions
- Architectural views
- Architectural patterns
- Application architectures

Architecture vs Design

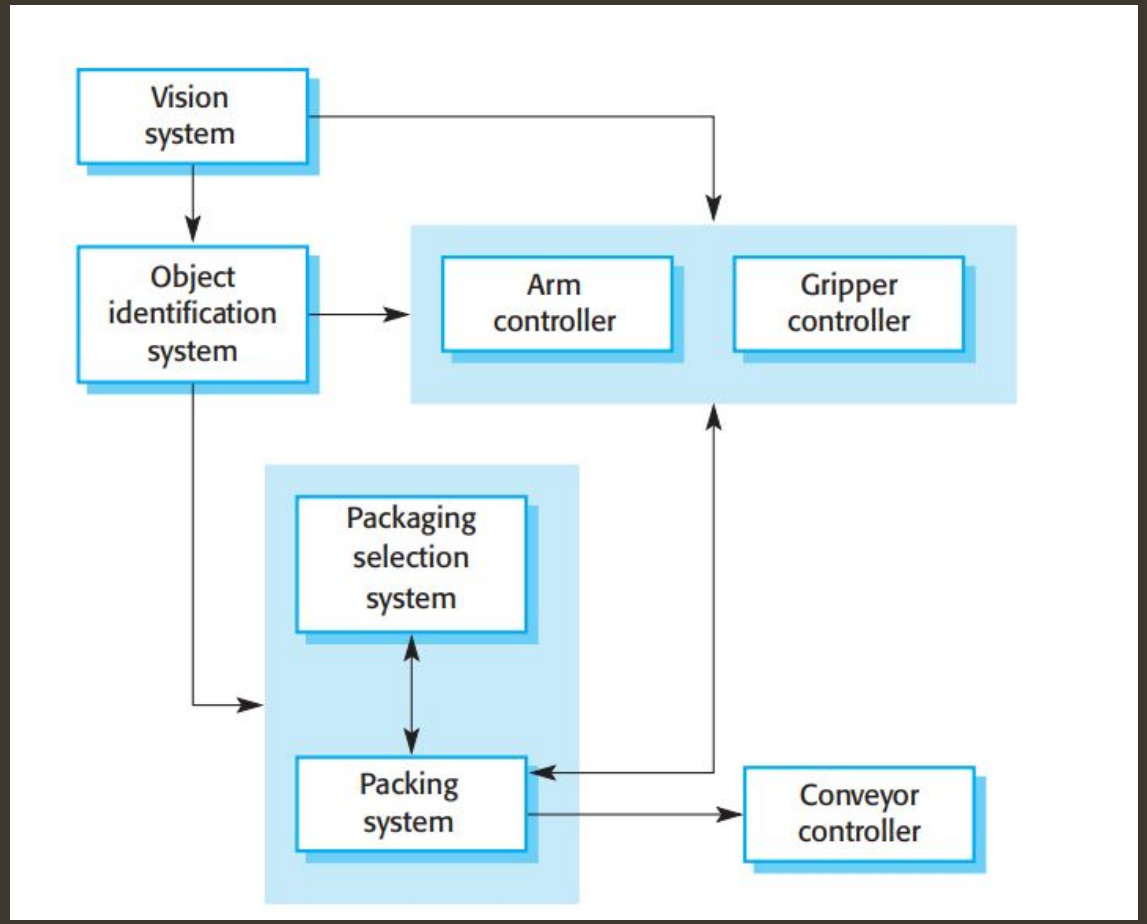
- There is a distinct difference between the terms architecture and design.
- A design is an instance of an architecture similar to an object being an instance of a class.
- For example, consider the client-server architecture. I can design a network-centric software system in many different ways from this architecture using either the Java platform (Java EE) or Microsoft platform (.NET framework). So, there is one architecture, but many designs can be created based on that architecture. Therefore, you cannot mix “architecture” and “design” with each other



Architectural design

- Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system.
- The architectural design starts then the developed software is put into the context.
- The information is obtained from the requirement model and other information collect during the requirement engineering.

Architecture Design



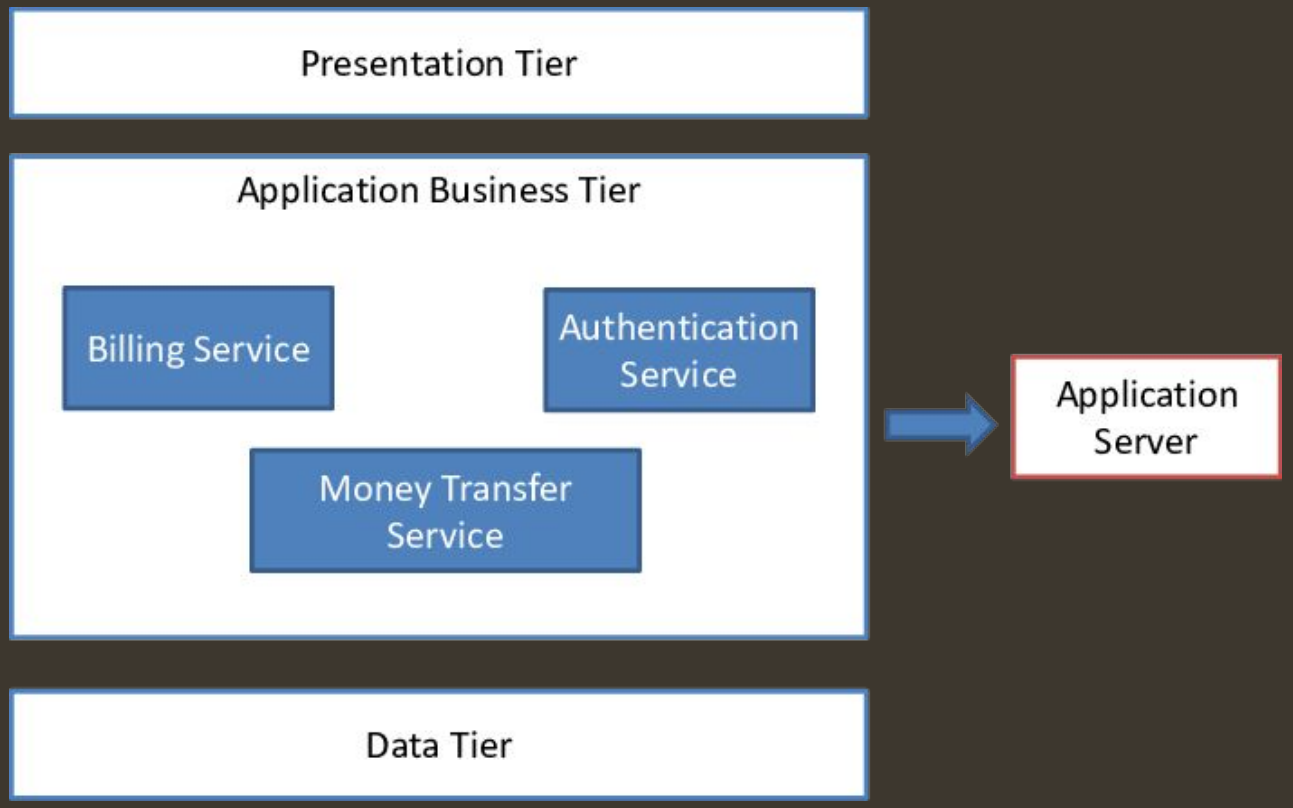
Agility and architecture

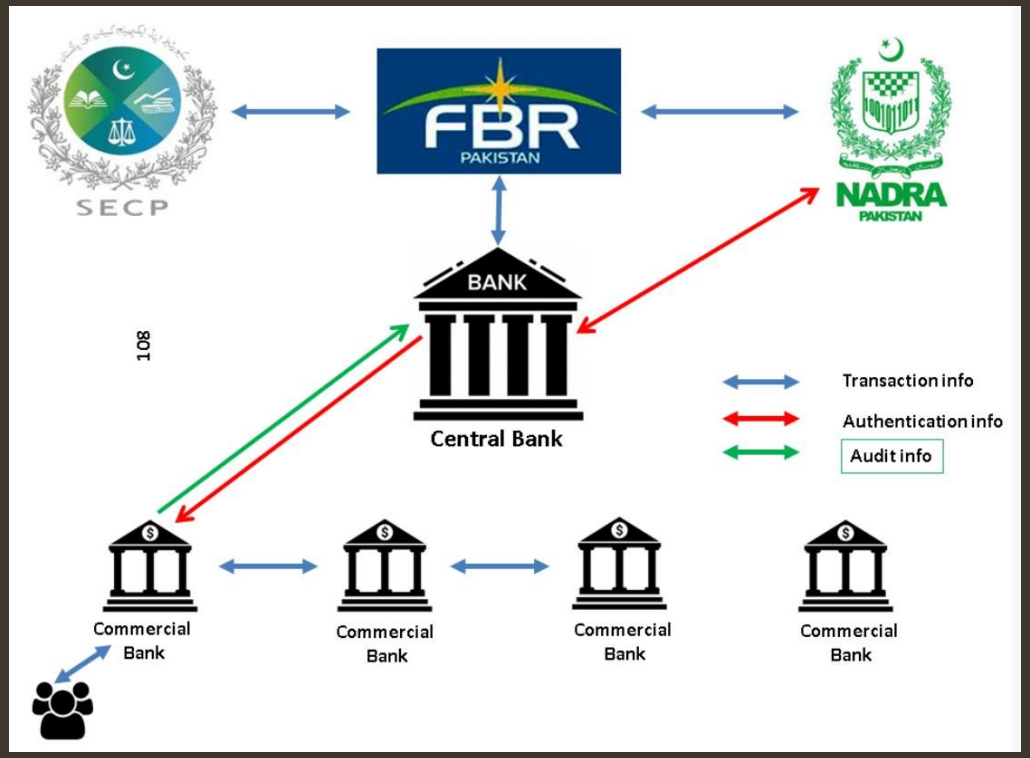
- It is generally accepted that an early stage of agile processes is to design an overall systems architecture.
- Refactoring the system architecture is usually expensive because it affects so many components in the system

Architecture level of 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. This chapter is mostly concerned with program architectures.
- ***Architecture in the large*** is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems may be distributed over different computers, which may be owned and managed by different companies.

Architecture level of abstraction





Advantages of explicit architecture

- “architecturally significant requirements” where they found that non-functional requirements had the most significant effect on the system’s architecture.
- Bass et al. (suggest that explicitly designing and documenting software architecture has three advantages):

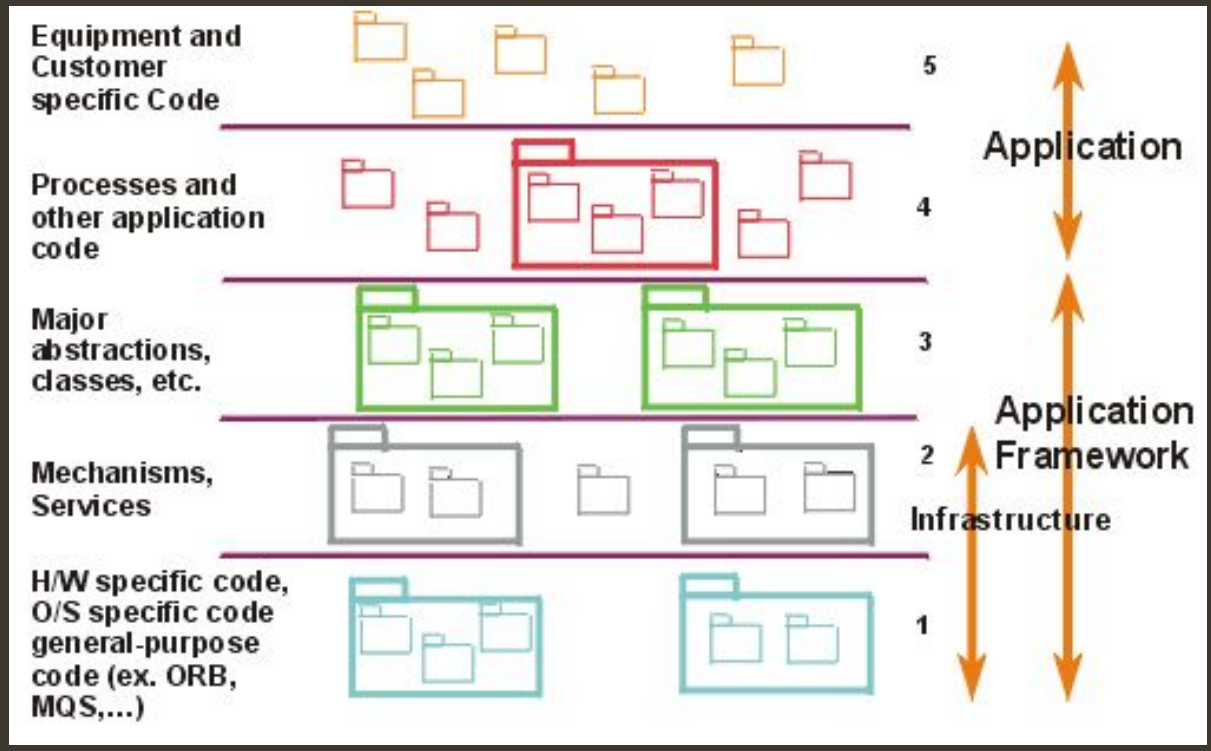
•Stakeholder communication
  •Architecture may be used as a focus of discussion by system stakeholders.

•System analysis
  •Means that analysis of whether the system can meet its non-functional requirements is possible.

•Large-scale reuse
  •The architecture may be reusable across a range of systems
  •Product-line architectures may be developed.

Architectural representations

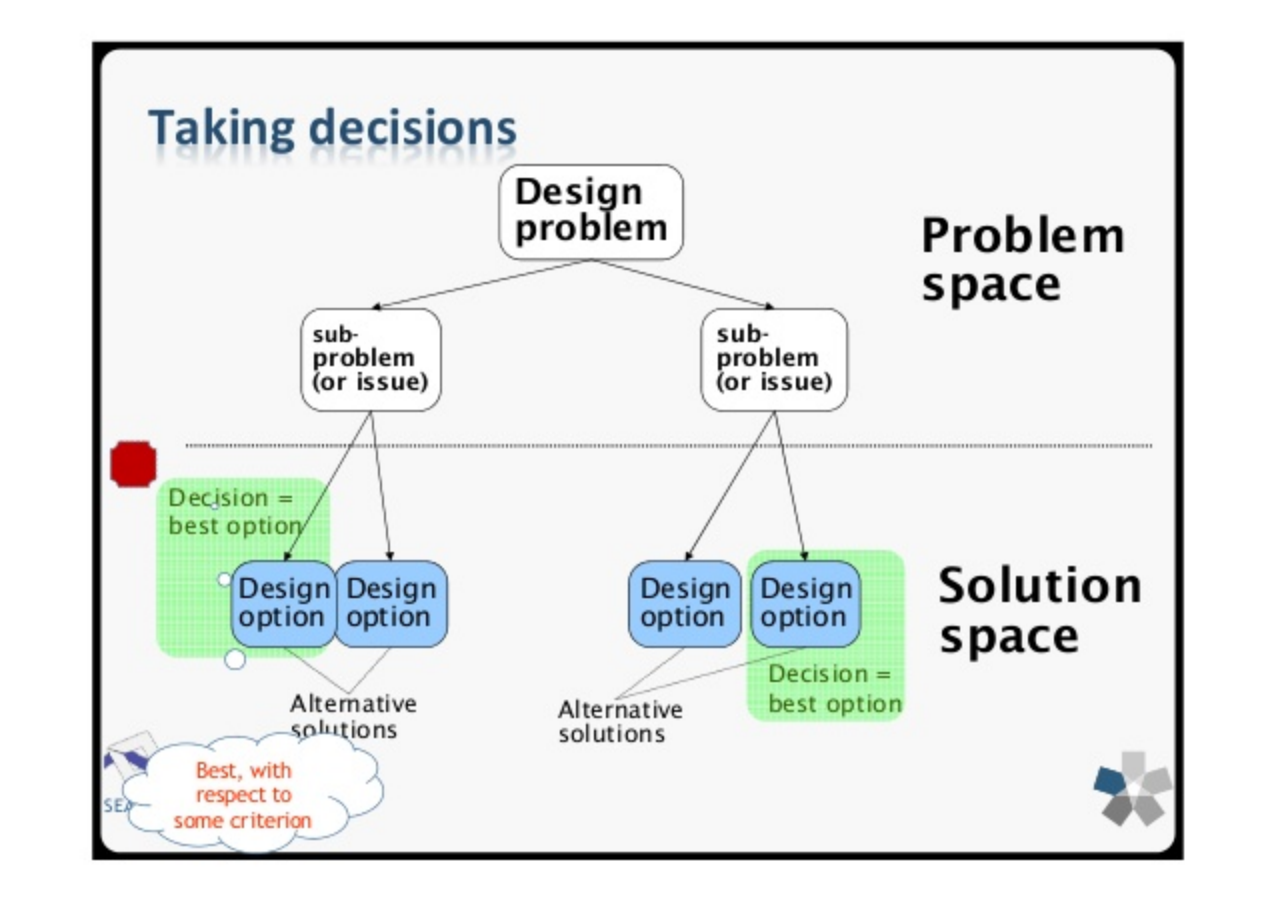
- Block diagrams present a high-level picture of the system structure
- Each box in the diagram represents a component. Boxes within boxes indicate that the component has been decomposed to subcomponents. Arrows mean that data and or control signals are passed from component to component in the direction of the arrows



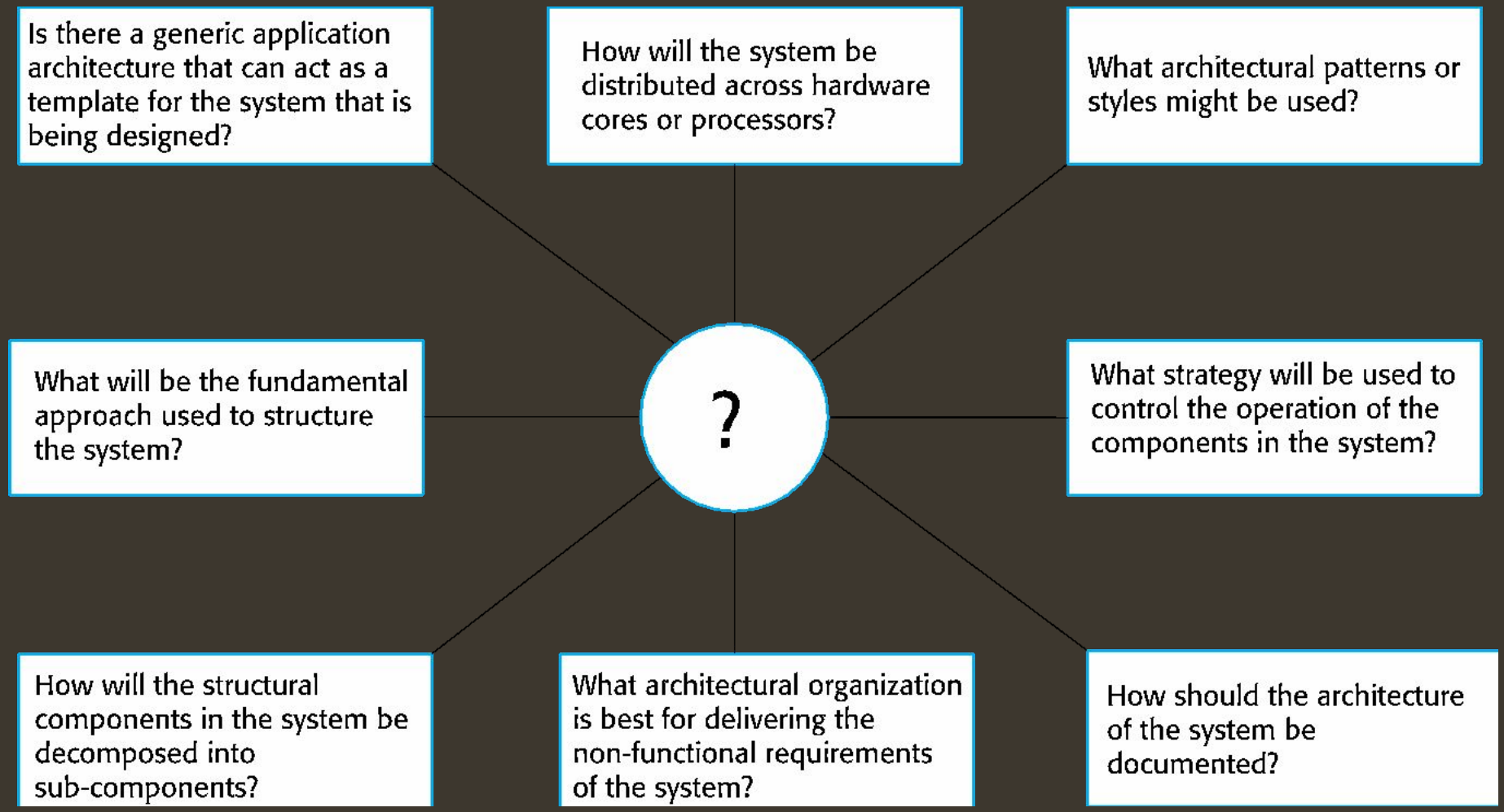
Class 20
01-March-2021

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



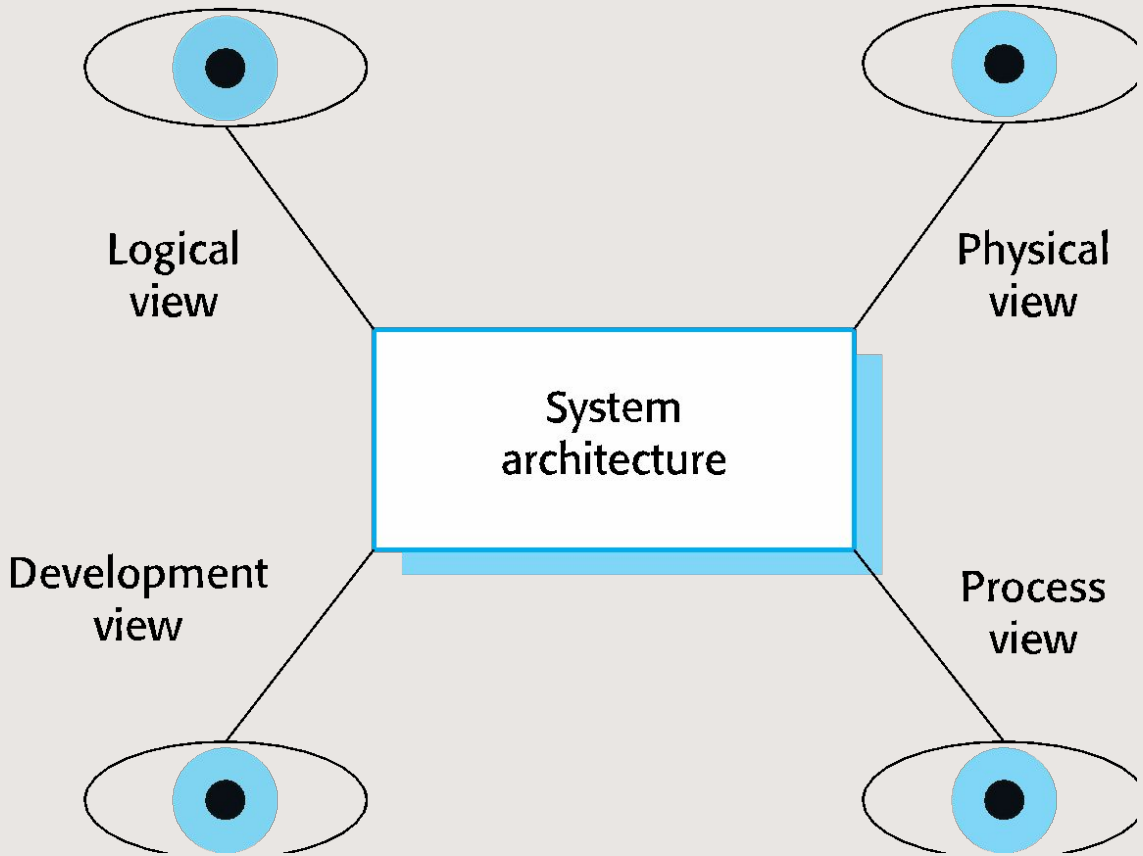
Architecture and system characteristics Quality Attributes

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

Architectural views

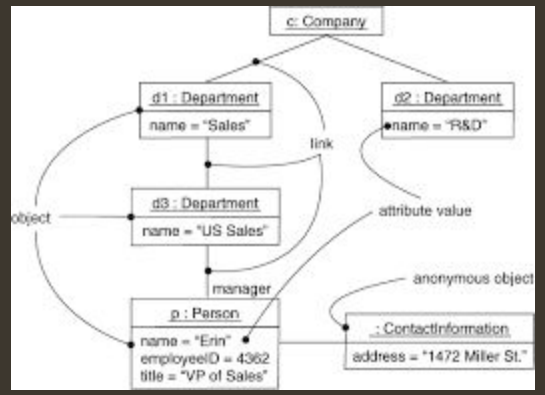
- What views or perspectives are useful when designing and documenting a system's architecture?
- What notations should be used for describing architectural models?
- It is impossible to represent all relevant information about a system's architecture in a single diagram, as a graphical model can only show one view or perspective of the system. ?
- for both design and documentation, you usually need to present multiple views of the software architecture.
- There are different opinions as to what views are required. Krutchen (Krutchen 1995) in his well-known 4 +1 view model of software architecture

Architectural views



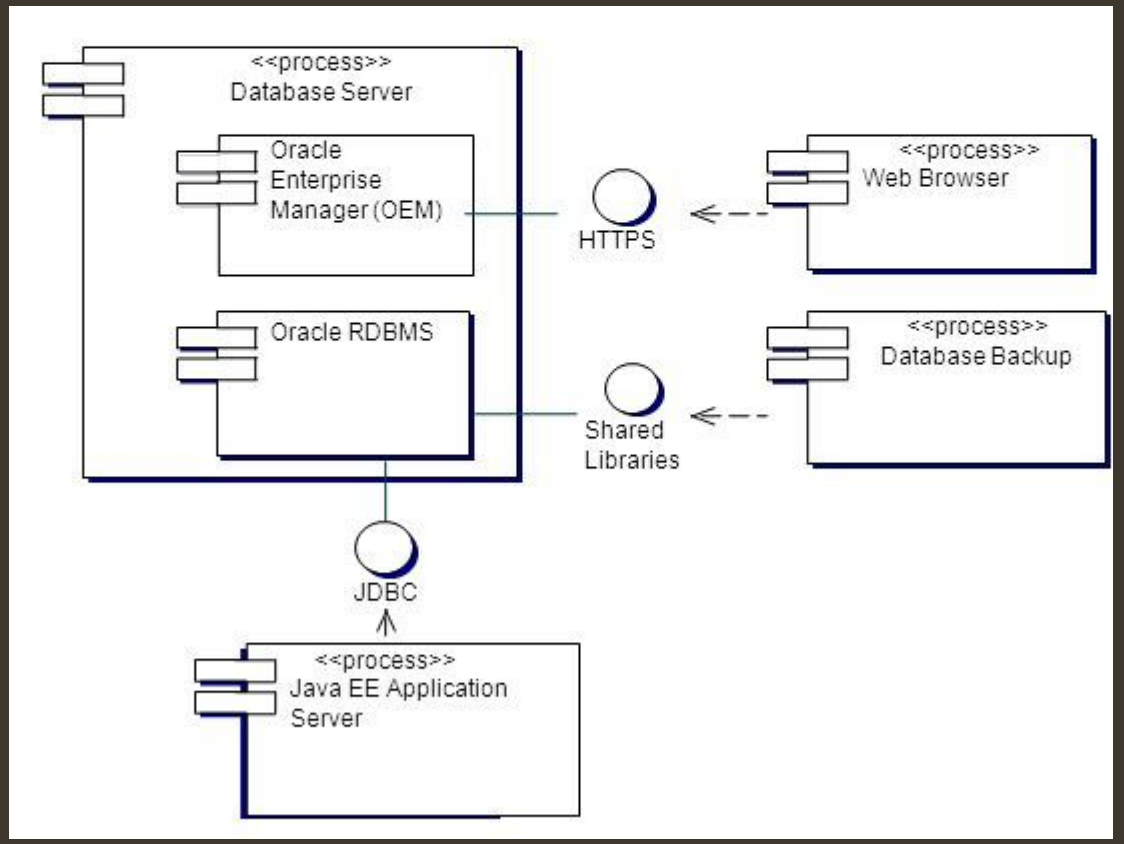
Logical View

- Logical view focus on functional requirement
- Show the key abstraction in system as object or object.
- Describes how the system is structured in terms of units of implementation. The elements are packages, classes, and interfaces.
- Viewer : Ender User
- The relationship between elements shows dependencies, interface realizations, part-whole relationships, and so forth.
  - Class Diagram
  - Sequence Diagram
  - Communication Diagram

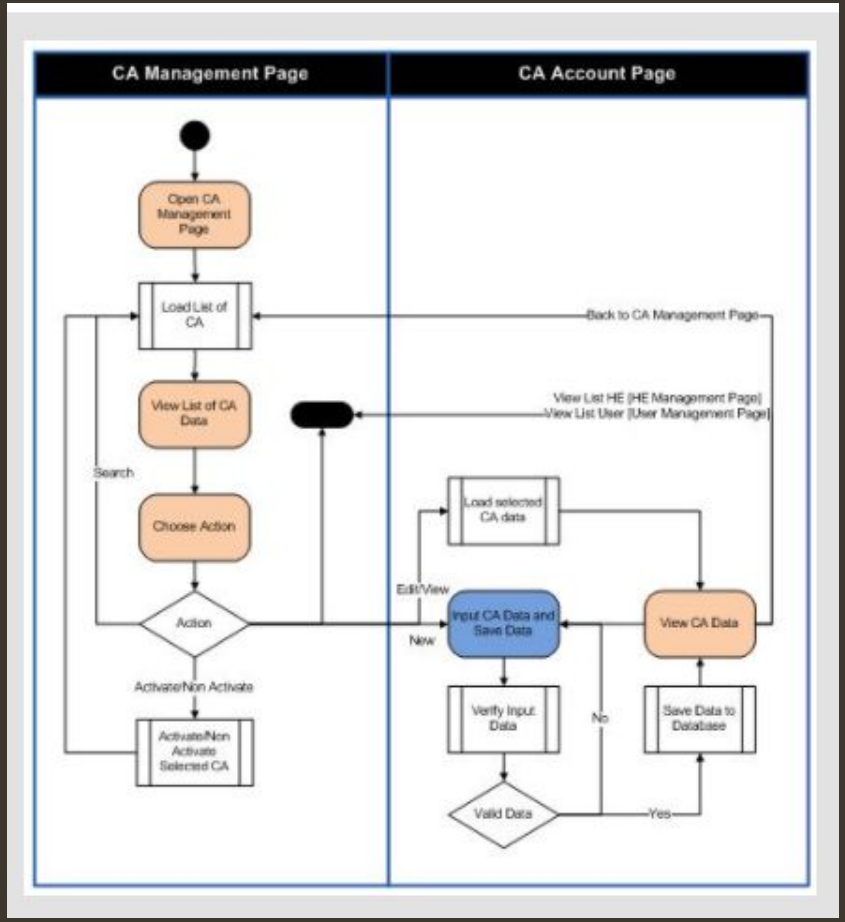


Process View

- How system is run
- Focus on achieving no-functional requirement
- which shows how, at runtime, the system is composed of interacting processes. This view is useful for making judgments about non-functional system characteristics such as performance and availability.
- Viewer: integrated

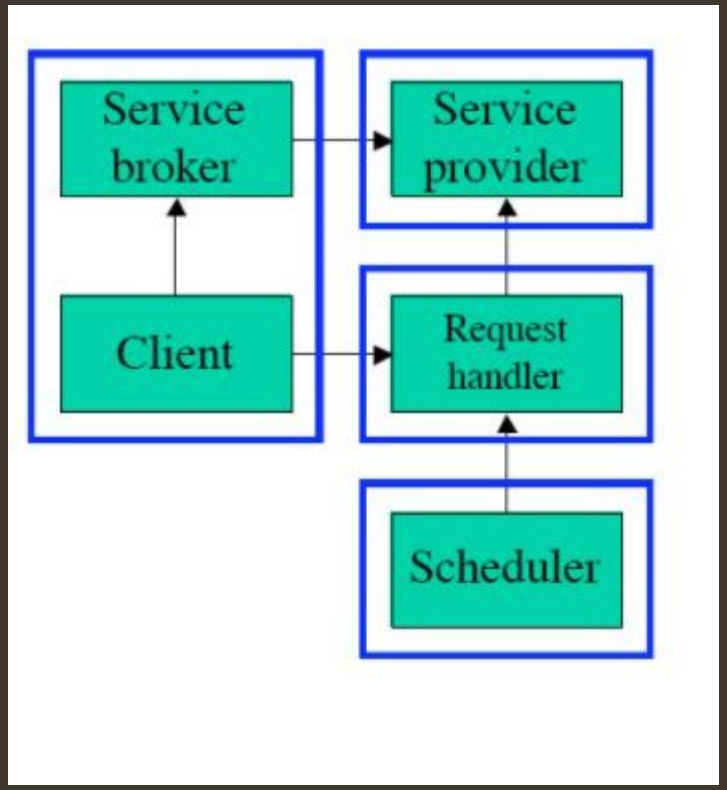


Process View



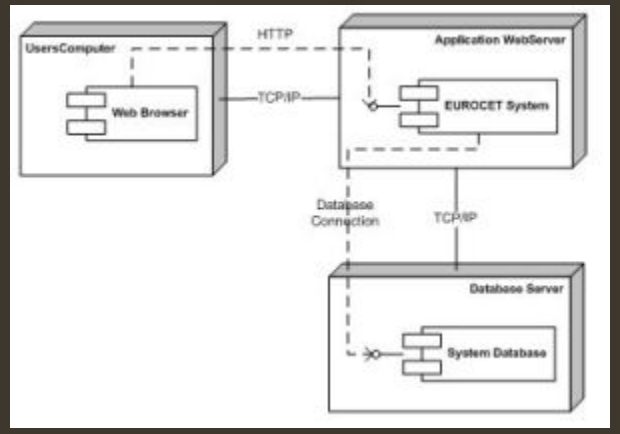
A development view

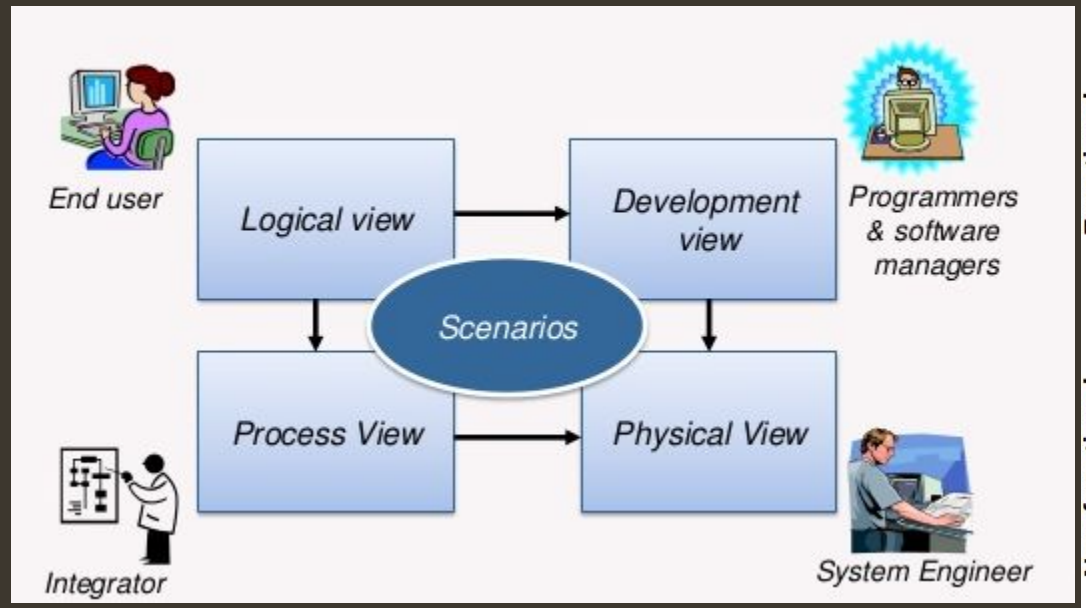
- It shows the decomposition of the system into different modules that will be implemented by different development teams and also represents the logical organization of the code.

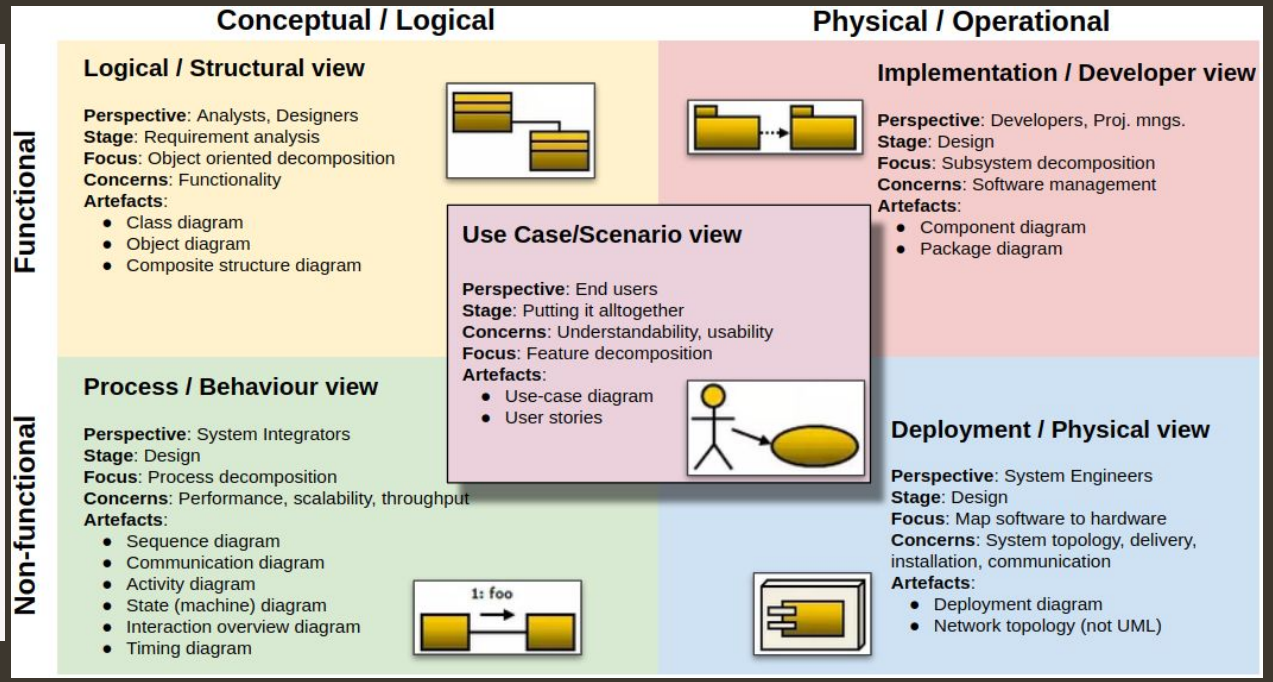


Implementation view

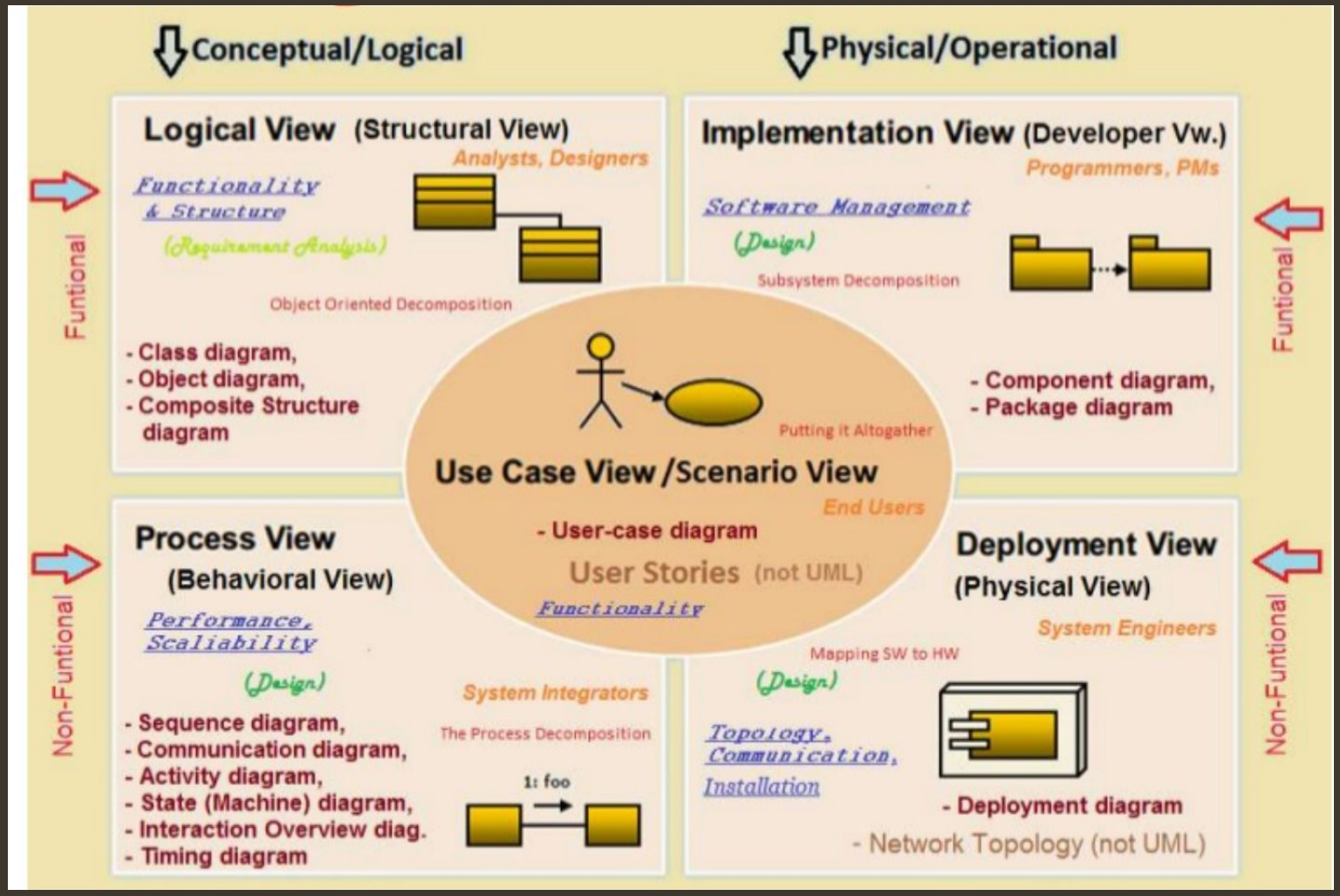
- Physical view is used for depicting how the software is deployed in data centers.
- For example in case of web based application you will show which web service you have used, no of servers and their role, and the software components installed on them. This view is useful for System engineers. The view below shows that two servers a database server and a web application server will be used in this EUROCET system.




End user
Logical view
Development view
Programmers & software managers
Scenarios
Process View
Physical View
Integrator
System Engineer


Conceptual / Logical
Physical / Operational
Functional
Non-functional
Logical / Structural view
Perspective: Analysts, Designers
Stage: Requirement analysis
Focus: Object oriented decomposition
Concerns: Functionality
Artefacts:
Class diagram
Object diagram
Composite structure diagram
Implementation / Developer view
Perspective: Developers, Proj. mngs.
Stage: Design
Focus: Subsystem decomposition
Concerns: Software management
Artefacts:
Component diagram
Package diagram
Use Case/Scenario view
Perspective: End users
Stage: Putting it alltogether
Concerns: Understandability, usability
Focus: Feature decomposition
Artefacts:
Use-case diagram
User stories
Process / Behaviour view
Perspective: System Integrators
Stage: Design
Focus: Process decomposition
Concerns: Performance, scalability, throughput
Artefacts:
Sequence diagram
Communication diagram
Activity diagram
State (machine) diagram
Interaction overview diagram
Timing diagram
1: foo
Deployment / Physical view
Perspective: System Engineers
Stage: Design
Focus: Map software to hardware
Concerns: System topology, delivery, installation, communication
Artefacts:
Deployment diagram
Network topology (not UML)

Summary


Conceptual/Logical
Physical/Operational
Logical View (Structural View)
Analysts, Designers
Functionality & Structure
(Requirement Analysis)
Object Oriented Decomposition
- Class diagram,
- Object diagram,
- Composite Structure diagram
Implementation View (Developer Vw.)
Programmers, PMs
Software Management
(Design)
Subsystem Decomposition
- Component diagram,
- Package diagram
Funtional
Funtional
Putting it Altogather
Use Case View /Scenario View
End Users
- User-case diagram
User Stories (not UML)
Functionality
Process View
(Behavioral View)
Performance, Scaliability
(Design)
System Integrators
The Process Decomposition
1: foo
- Sequence diagram,
- Communication diagram,
- Activity diagram,
- State (Machine) diagram,
- Interaction Overview diag.
- Timing diagram
Deployment View
(Physical View)
System Engineers
Mapping SW to HW
(Design)
Topology, Communication, Installation
- Deployment diagram
- Network Topology (not UML)
Non-Funtional
Non-Funtional

Architecture Pattern and Design Pattern

- Software architecture is responsible for the skeleton and the high-level infrastructure of software,
- whereas software design is responsible for the code level design such as, what each module is doing, the classes scope, and the functions purposes, etc.
- Generally, the architecture and design both explains the **idea** but **architecture** focus on the **abstract view** of idea while **design** focus on the **implementation view** of idea.

Design Patterns:

It solves reoccurring problems in the software Design.

- **Creational Patterns**: It focus on how to instantiate an object or group of related objects.
    1. Abstract Factory
    2. Builder
    3. Factory Method
    4. Prototype
    5. Singleton pattern
- **Structural Patterns**: Identifying a simple way to realize relationships among Entities.
    1. Adapter
    2. Bridge
    3. Decorator
    4. Facade
    5. Proxy pattern

- Behavioral Patterns: It identify common communication patterns between objects and realize these patterns.
    - Command
    - Interpreter
    - Iterator
    - Mediator
    - Null Object
    - Observer

Architectural Pattern

- Patterns are a means of representing, sharing and reusing knowledge. An architectural pattern is a **stylized description of a 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.
- Patterns help you build on the collective experience of skilled software engineers.

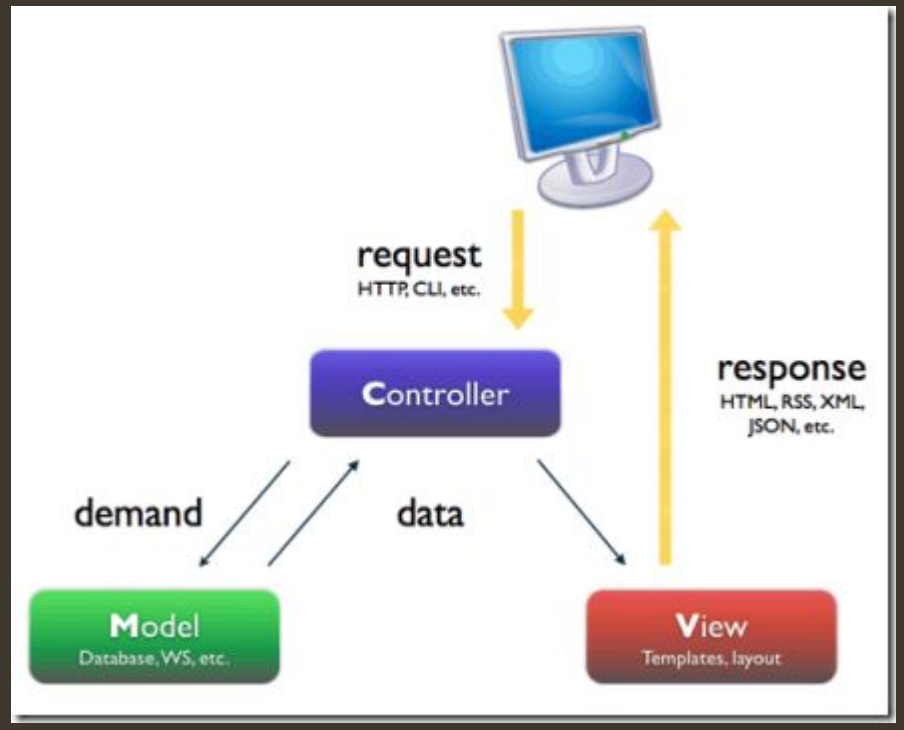
The architectural patterns address various issues such as :

- Performance
- High availability
- Scalability
- Security
- Maintainability
- Testing
- Deployment
- Technology Stack

#Examples:

- MVC Pattern
- Model View Presenter (MVP)
- Layered pattern (N-tier Architecture)
- Master-Slave Pattern
- REST (Architecture Style)
- Monolithic architecture
- Micro service architecture
- Event-driven architecture (EDA)
- Service-Oriented architecture (SOA)

MVC


request
HTTP, CLI, etc.
Controller
response
HTML, RSS, XML, JSON, etc.
demand
data
Model
Database, WS, etc.
View
Templates, layout

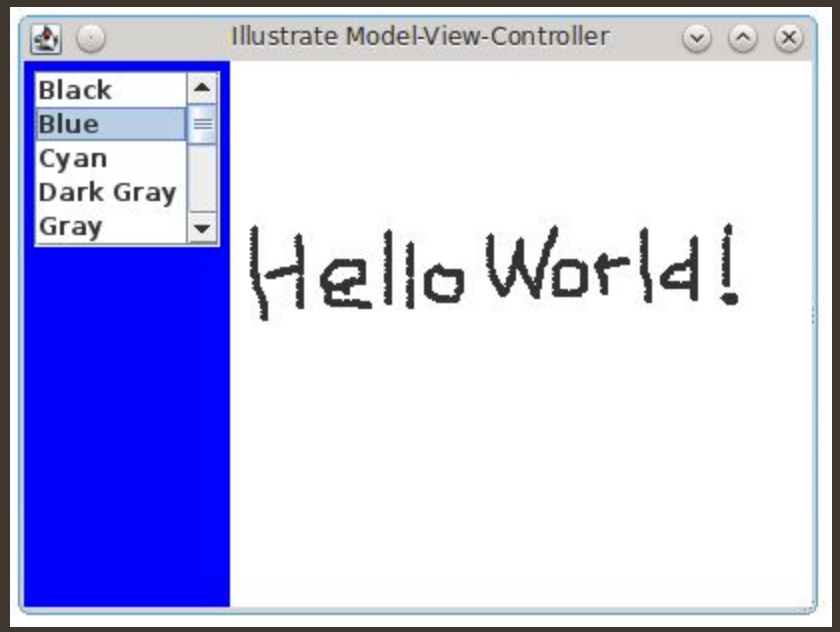
The Model-View-Controller (MVC) pattern

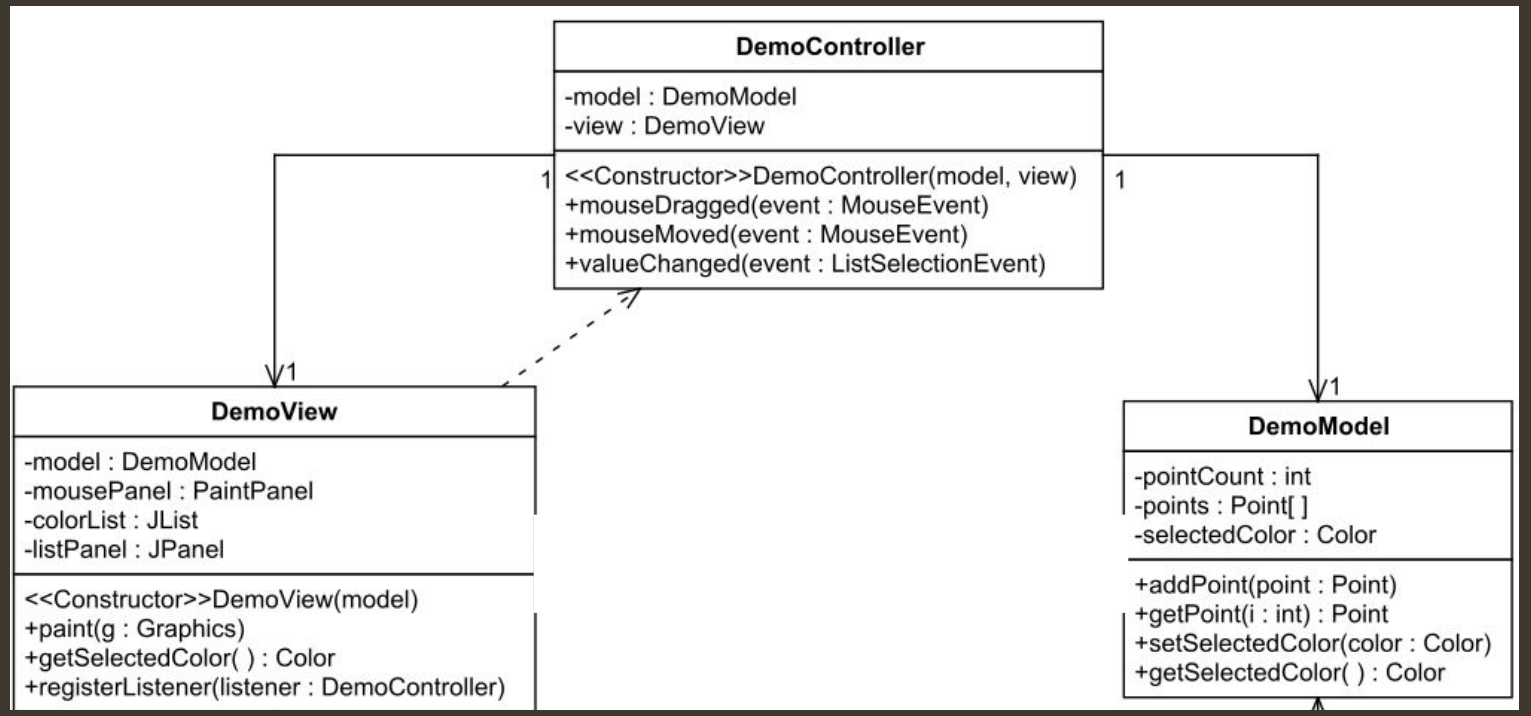
- Patterns may be described in a standard way using a mixture of narrative description and diagrams.

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 6.5.
Example	Figure 6.6 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	May involve additional code and code complexity when the data model and interactions are simple.

Architectural pattern should describe a system organization that has been successful in previous systems. It should include information on when it is and is not appropriate to use that pattern, and details on the pattern's strengths and weaknesses.

Demo Example





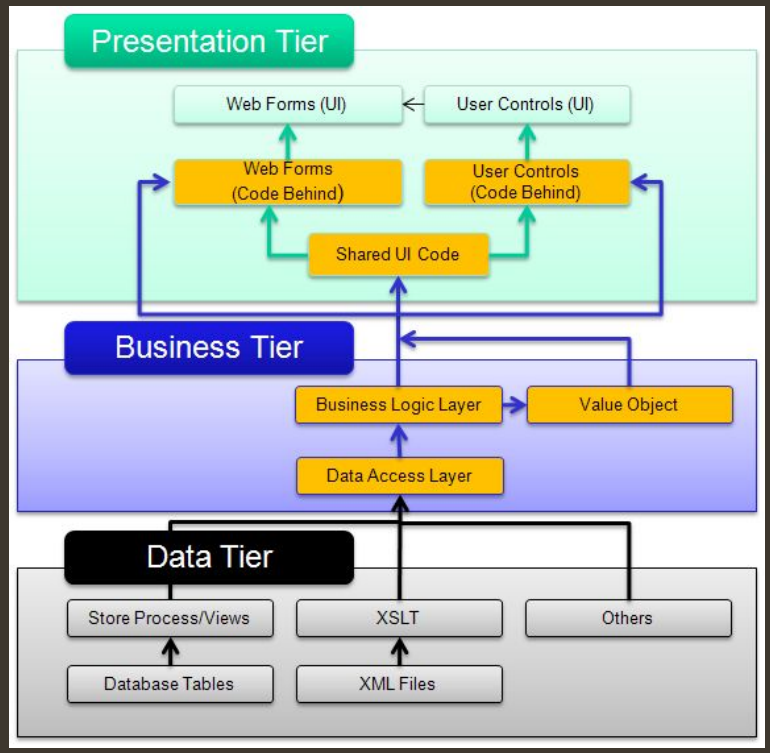
1. A DemoModel object has instance variables for storing an array of Points and a color.
2. DemoController implements the MouseMotionListener interface (mouseDragged and mouseMoved method) and the ListSelectionListener interface (valueChanged method).
3. DemoView is a subclass of JFrame.

Layered architecture patterns

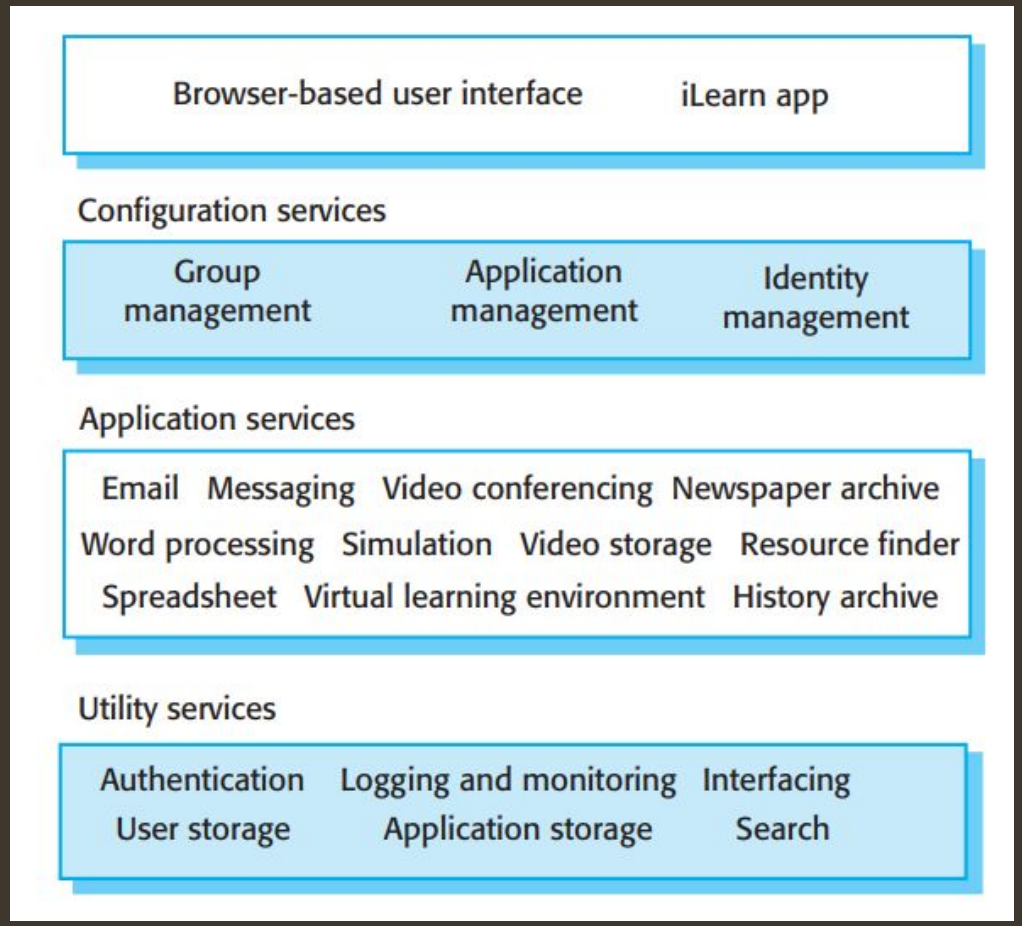
- Layered architecture patterns are n-tiered patterns where the components are organized in horizontal layers.
- This is the traditional method for designing most software and is meant to be self-independent.
- separates elements of a system, allowing them to change independently.
- For example, adding a new view or changing an existing view can be done without any changes to the underlying data in the model.
- This layered approach supports the incremental development of systems. As a layer is developed, some of the services provided by that layer may be made available to users

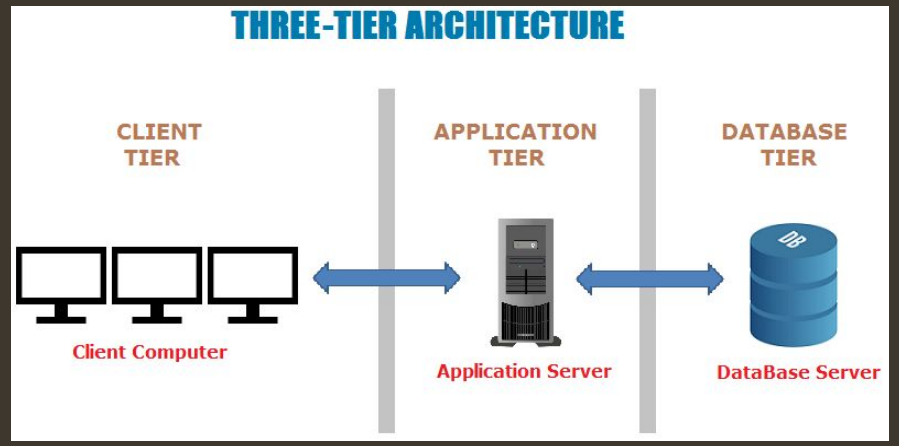
The four standard layers are described as follows.

- *Presentation Layer*
  This layer contains all user interfaces exposed to a user. It may provide different types of user interfaces, namely web, desktop and native mobile applications.
- *Business Logic Layer*
  This layer handles all business logic, validations and business processes.
- *Data Access Layer*
  This layer is responsible for interacting with a database. It is also known as the persistence layer.
- *Data Store Layer*
  This layer is the actual data store for the application.



The architecture of the iLearn system

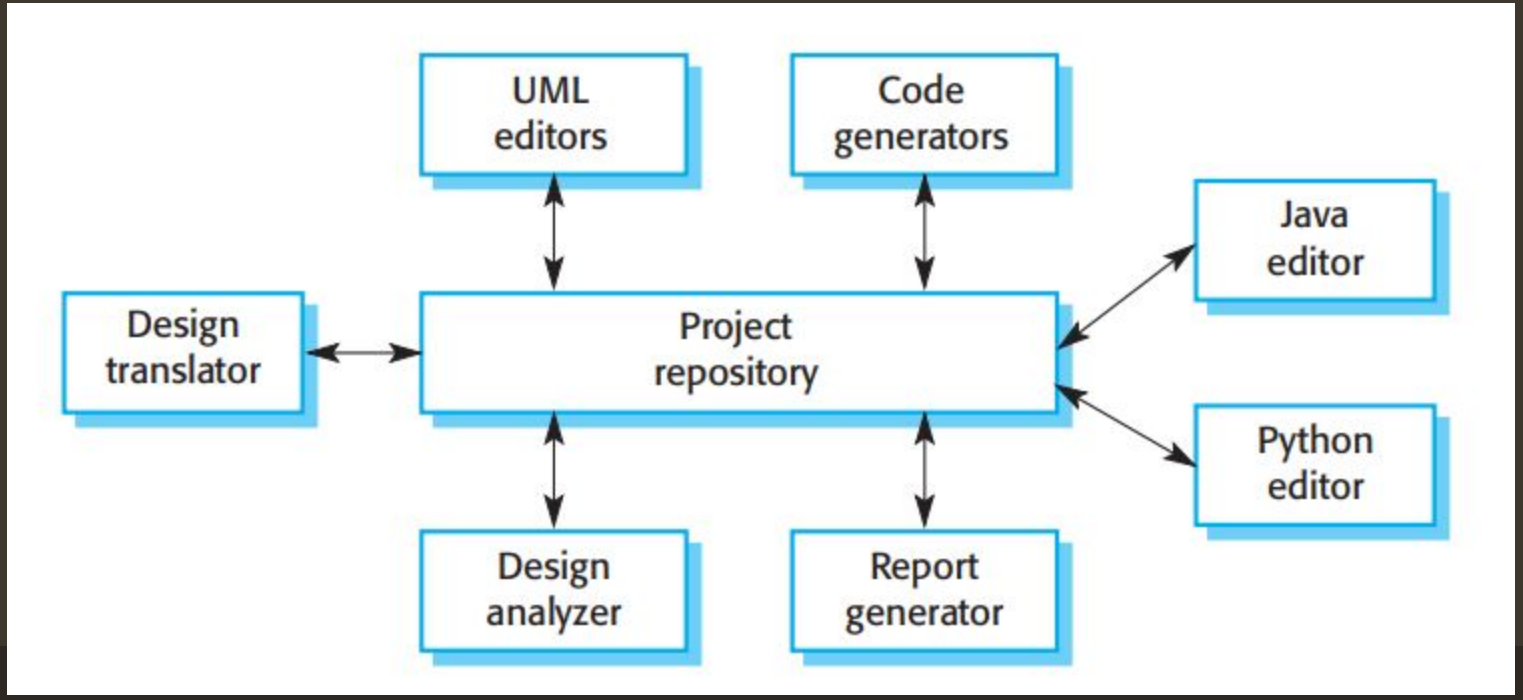




https://www.ecanarys.com/Blogs/ArticleID/76/3-Layered-Architecture

Repository architecture

- A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
- This model is therefore suited to applications in which data is generated by one component and used by another.



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 6.11 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.

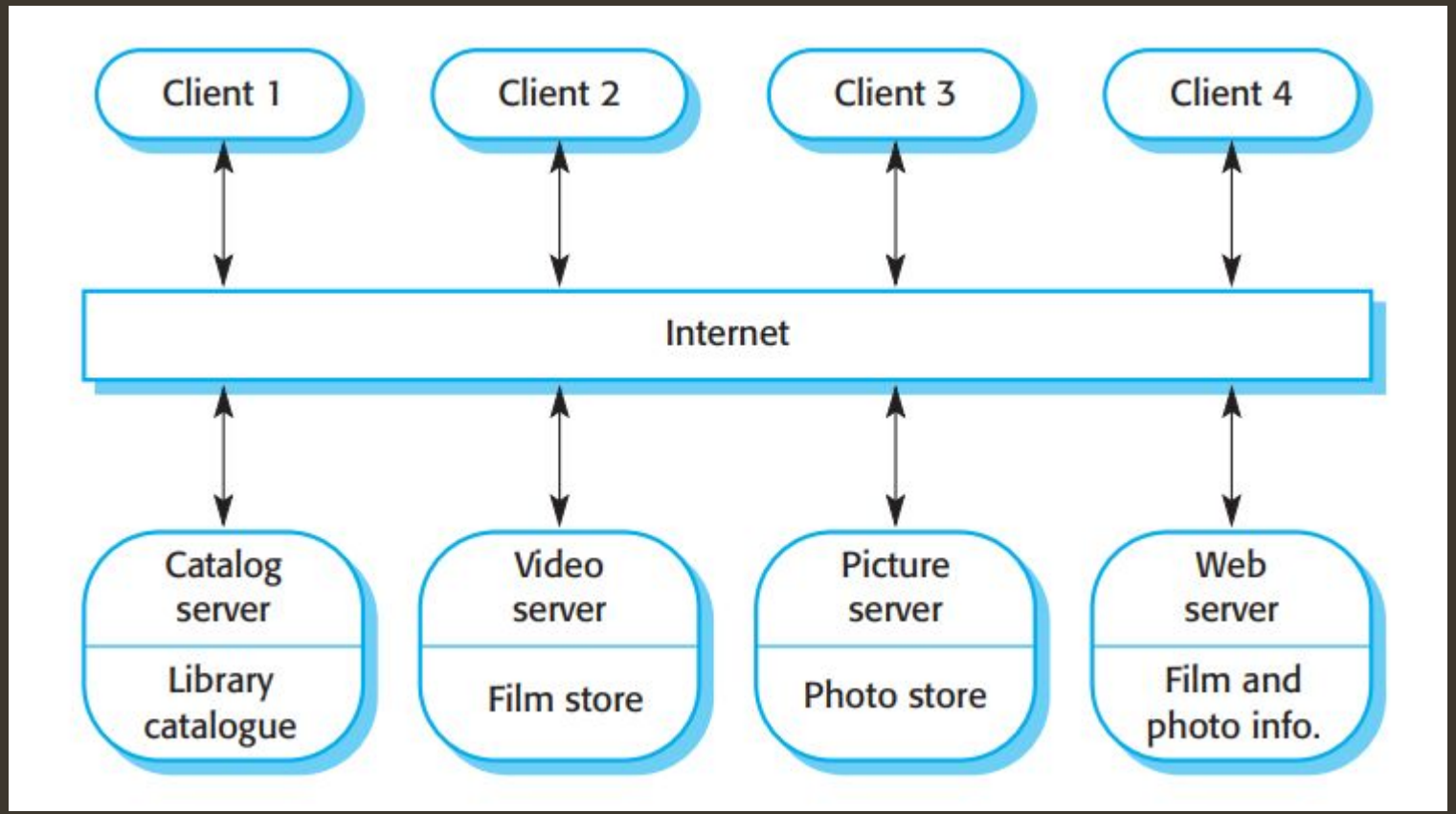
Client–server architecture

- Client-server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of
- architecture has one or more client computers connected to a central server over a network or
- internet connection. Client-server architecture is also known as a networking computing
- model or client-server network because all the requests and services are delivered over a
- network.

Client–server architecture

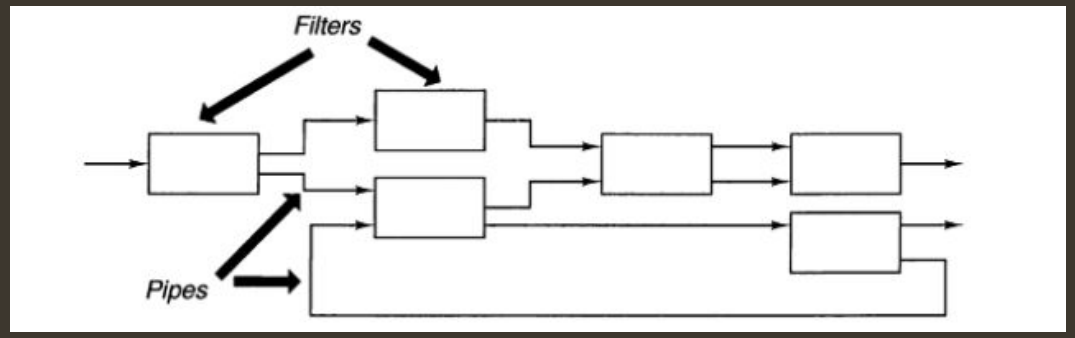
Name	Client–server
Description	In a client–server architecture, the system is presented as a set of services, with each service delivered by a separate server. Clients are users of these services and access servers to make use of them.
Example	Figure 6.13 is an example of a film and video/DVD library 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 and so is susceptible to denial-of-service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. Management problems may arise if servers are owned by different organizations.

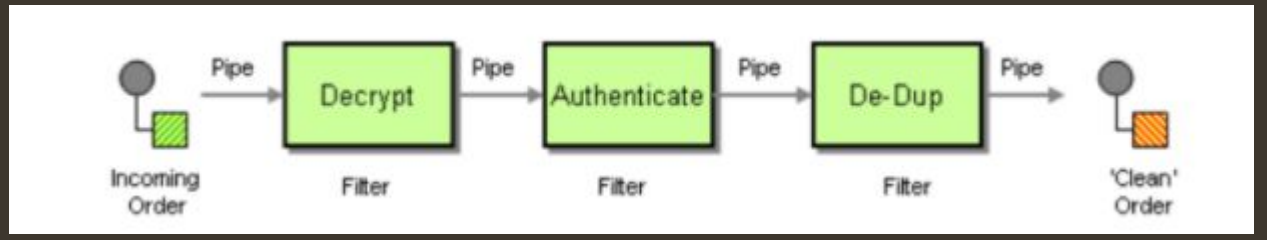
Example


Client 1
Client 2
Client 3
Client 4
Internet
Catalog server
Library catalogue
Video server
Film store
Picture server
Photo store
Web server
Film and photo info.

Pipe and filter architecture

- This is a model of the runtime organization of a system where functional transformations process their inputs and produce outputs.
- pipe and Filter is a simple architectural style that connects a number of components that process a stream of data, each connected to the next component in the processing pipeline via a **Pipe**.





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	Figure 6.15 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 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 architectural components that use incompatible data structures.