CS335FZ, 2021-2022

Architectural Design

Dr. Dapeng Dong

What is Software System Architecture?

"The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both."

-- Bass, L. Clements, P. and Kazman, R. Software architecture in practice, 3ed, 2013.

"The architecture of a system is the set of fundamental concepts or properties of the system in its environment, embodied in its elements, relationships, and the principles of its design and evolution."

-- Rozanski, N. and Woods, E., 2012. Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley.

"Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system."

-- Sommerville, I., 2016. Software engineering., 10th Edition. Pearson Education.

The Importance of Software Architecture

- Enhancing communication among stakeholders
- Allowing the architect and project manager to estimate the cost and schedule
- Defining constrains on subsequent implementation
- Containing the most fundamental and hard-to-change design decisions
- Creating transferable and reusable model for software products
- Facilitating system analysis

Architecture is an Abstraction

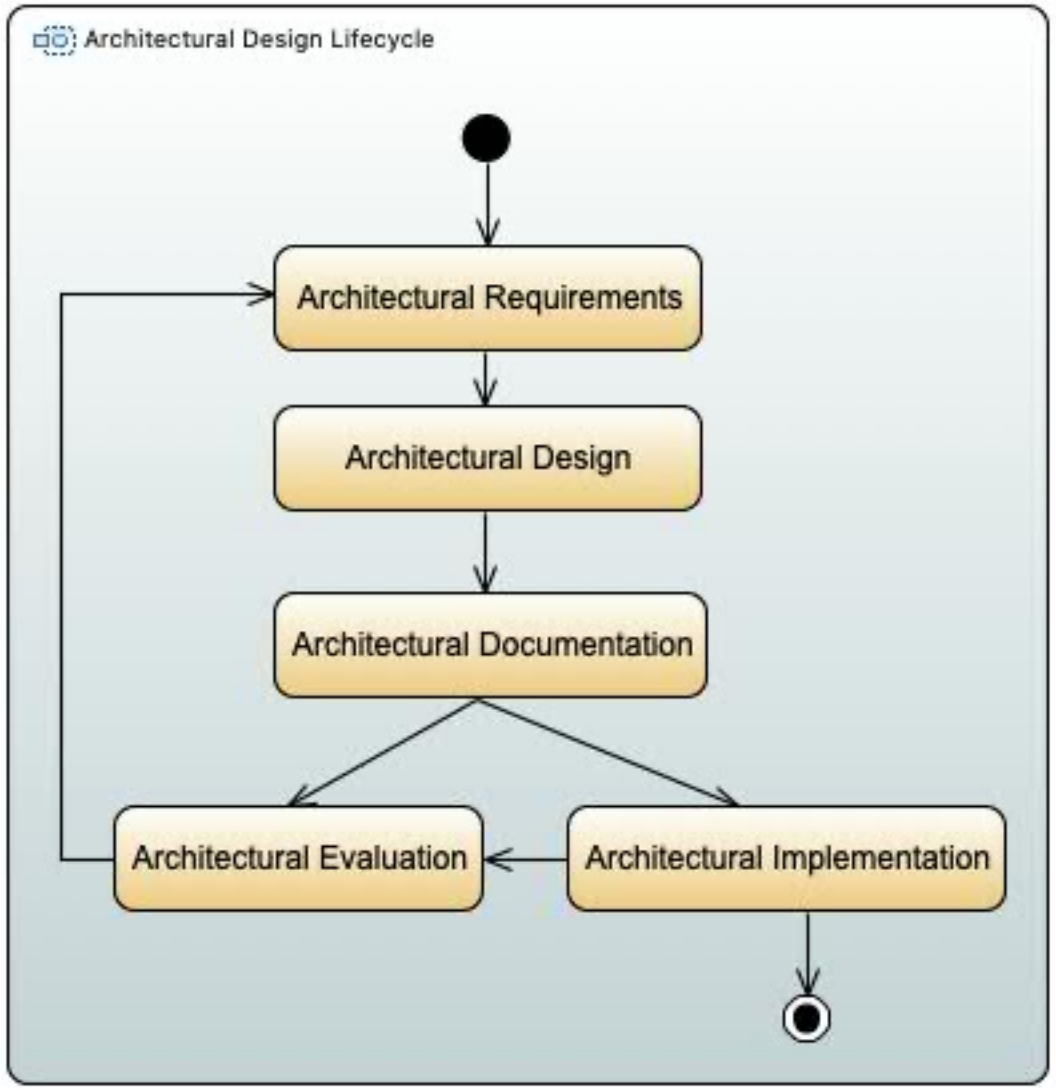
- In all modern systems, elements interact with each other by means of interfaces
- Interfaces partition details about an element into public and private parts
  - Private: details in internal implementation
  - Public: how elements are arranged, interacted with others, composed, and its properties that support reasoning of the design.

Private details of element are NOT architectural!

- Levels of abstraction
  - Architecture in the small
    - Architecture of individual programs
    - Mostly concerned with how an individual program can be decomposed into components
    - Individual components implement the functional system requirements
  - Architecture in the large
    - Architecture of complex systems of distributed components
    - Mostly concerned with how distributed components can be composed to form a new system

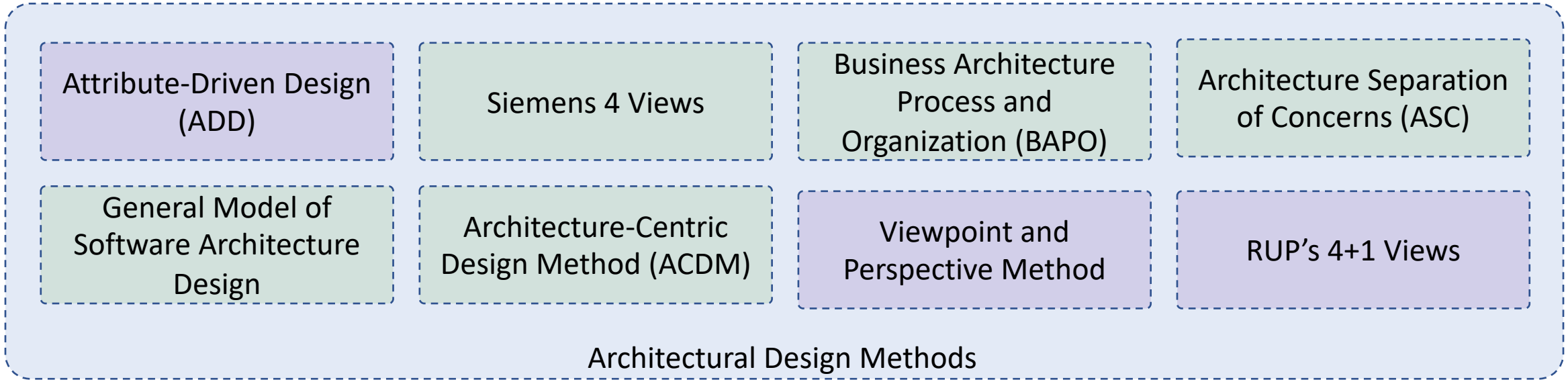
Architecture Design Lifecycle

- Among all requirements there are a few that have special importance for the architecture, known as Architecturally Significant Requirements (ASRs):
  - E.g., the most important functionality of the system, the constraints and the quality attributes such as high performance and high availability, etc.
- Design is a translation, from requirements to solutions, which can be structures composed of code, frameworks, and components.
- Preliminary documentation (*sketches*) of the structures should be created as part of architectura design.
- If the project under development is non-trivial, then the design should be evaluated to ensure that the decisions made are appropriate to address the ASRs.
- An architect's responsibility during implementation is to ensure *conformance* of the code to the design.

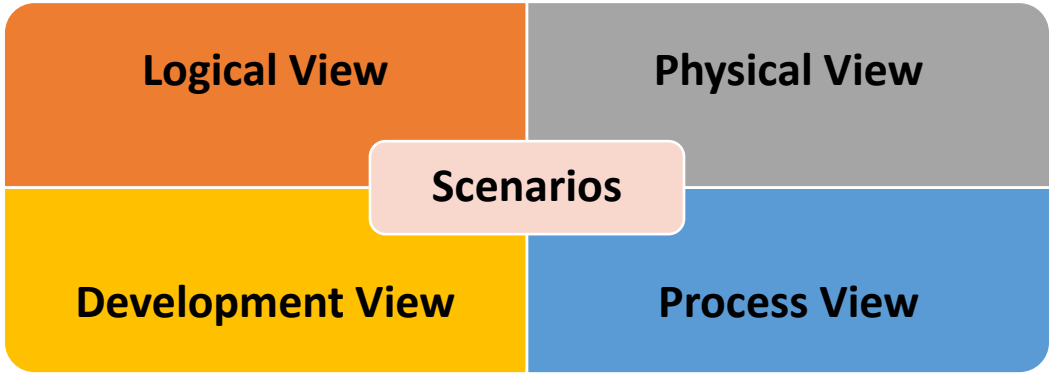


Principled Methods

- How do we actually perform a design?
- Performing design to *ensure* that the business requirements are satisfied requires a principled method.
- A method provides guidance.



Architectural Views (4+1)

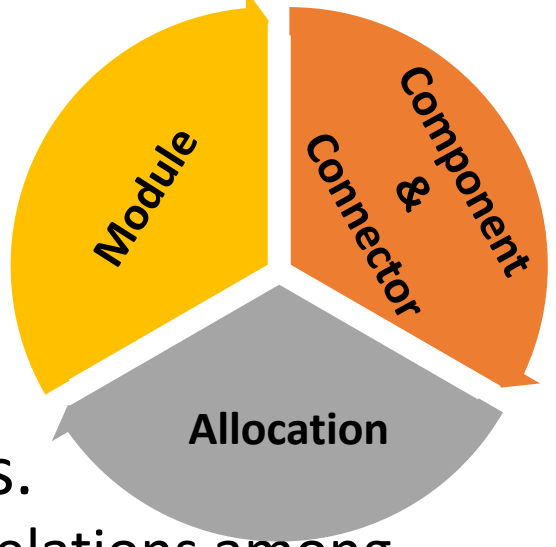


The 4+1 View Model

- A *logical view* shows the key abstractions in the system as objects or object classes.
  - Useful for relating system requirements to elements in a logical view
- A physical view shows the system hardware and how software components are distributed in the system.
  - Useful for planning a system deployment
- A *process view* shows how the system is composed of interacting processes at run time.
  - Useful for making judgments about non-functional system characteristics
- A development view shows how the software is decomposed for development.
  - Useful for allocating works and planning development
- A Scenario is used to illustrate and validate the 4 views.

Reference (The 4+1 Model): Kruchten, P.B., 1995. The 4+ 1 view model of architecture. IEEE software, 12(6), pp.42-50.

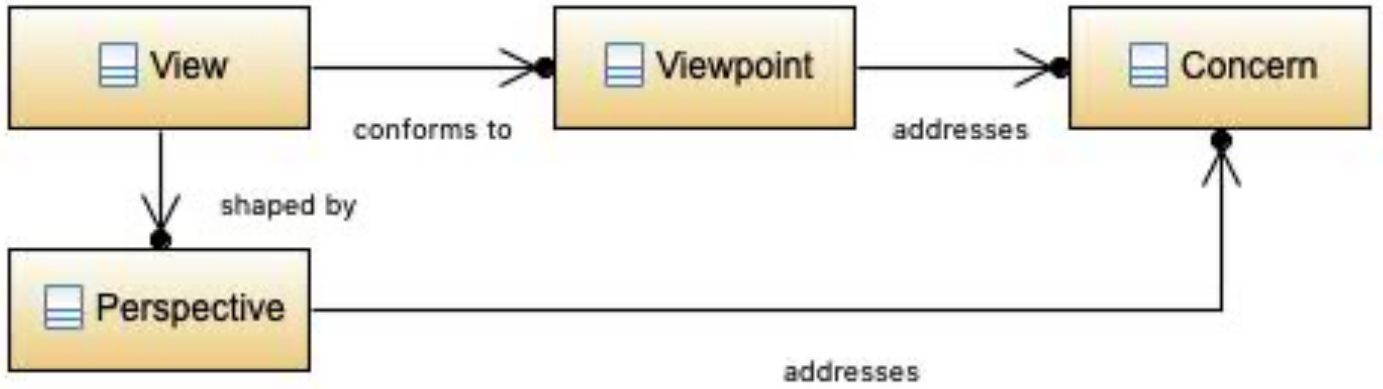
Structures and Views (ADD)



- A view is a representation of a set of architectural elements.
  - A view consists of a representation of a set of elements and the relations among them.
- A structure is the set of elements itself, as they exist in software or hardware.
  - Module structures: how the system is to be structured as a set of code or data units that must be constructed or procured.
  - Component-and-connector structures: how the system is to be structured as a set of elements that have runtime behavior (components) and interactions (connectors).
  - Allocation structures show the relationship between the software elements and elements in its operational environments.
- Architects design structures. They document views of those structures.

Reference (Structures & Views): Bass, L. Clements, P. and Kazman, R. Software architecture in practice, 3ed, 2013.

Viewpoints & Perspectives



"A Viewpoint is a collection of patterns, templates, and conventions for constructing one type of view."

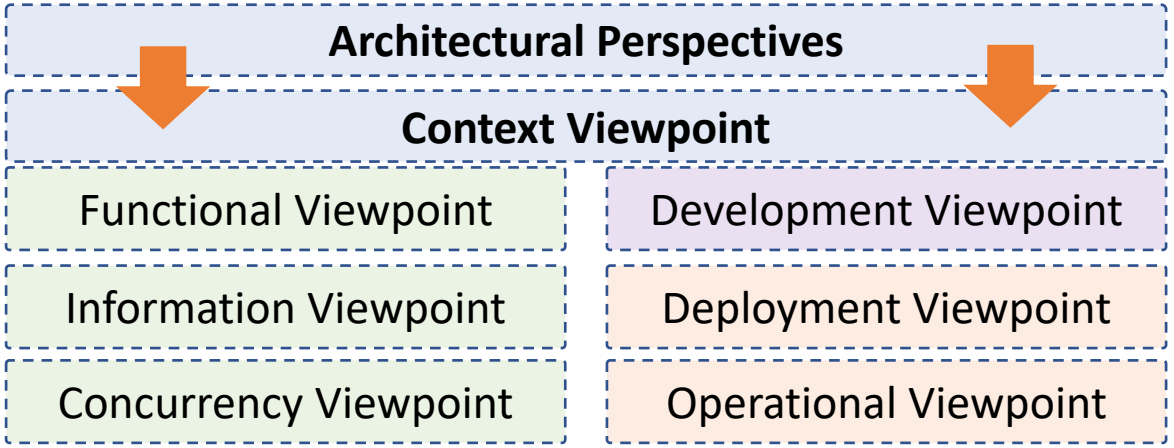
"A View is a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders."

"An architectural perspective is a collection of architectural activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system's architectural views."

Rozanski, N. and Woods, E., 2012. Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley.

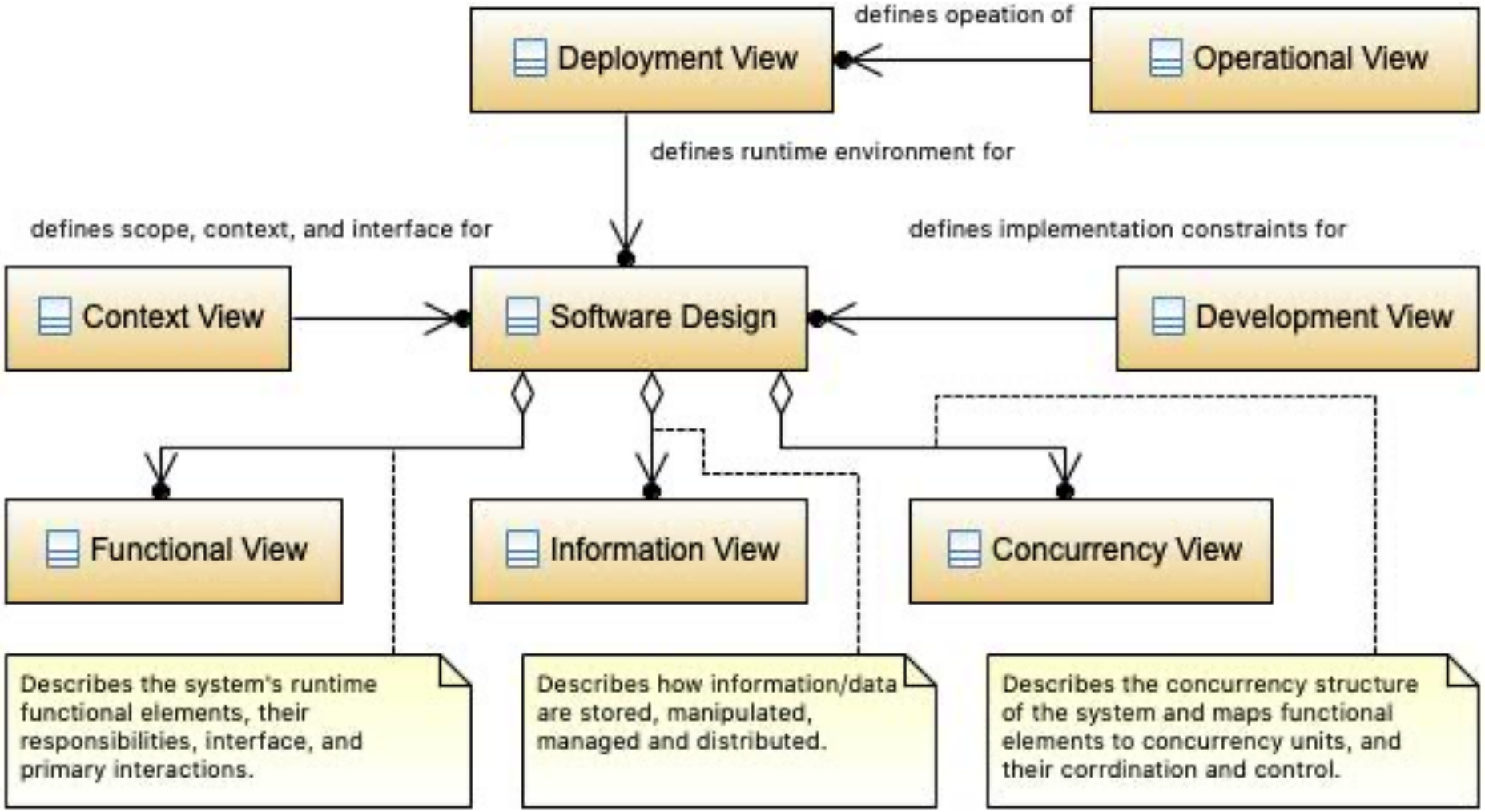
Viewpoints

A *viewpoint* provides a template for the construction of a *view*.



- *Context viewpoint:* describes the relationships, dependencies and interactions between the system and its environment.
- *Functional viewpoint:* describes the system's functional elements, their responsibilities, interfaces, and interactions.
- *Information viewpoint:* describes how the system stores, manipulates, manages, and distribute information.
- *Concurrency viewpoint:* describes the concurrency structure of the system and maps functional element to concurrency units.
- *Development viewpoint:* describes the architecture that supports the software development process.
- *Deployment viewpoint*: describes how the system will be installed and deployed in its operational environment with associated dependencies.
- *Operational viewpoint*: describes how the system will be operated, managed and supported

View Relationships



Architectural Patterns

- Architectural elements can be composed in ways that solve particular problems.
  - The compositions have been found useful over time, and over many different domains
  - They have been documented and disseminated.
  - These compositions of architectural elements, called ***architectural patterns***.
  - Patterns provide packaged strategies for solving a particular kind of problem.
- An architectural pattern delineates the element types and their forms of interaction used in solving the problem.
  - Context, problem, and solutions

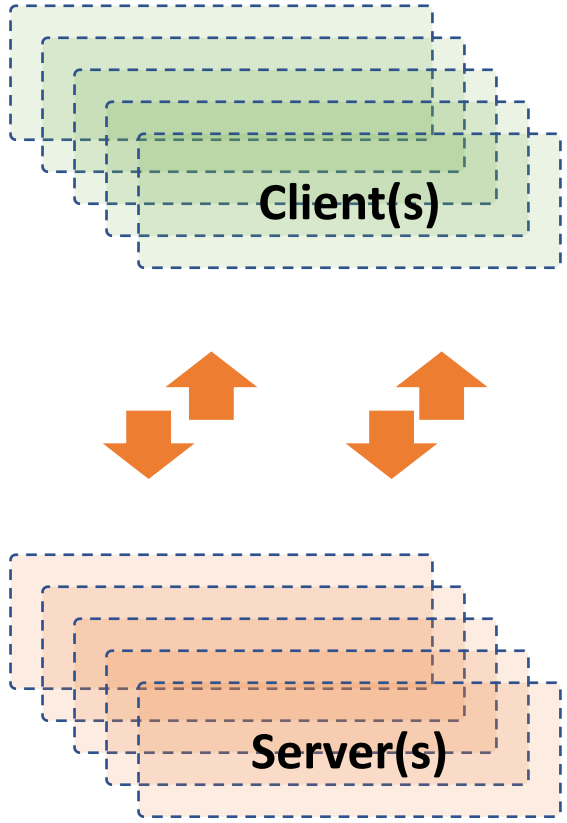
Reading (Architectural Style): Mary, S. and David, G., 1996. Software architecture: perspectives on an emerging discipline. Prentice-Hall.

Client-Server Architecture

- ***Context:*** *There are shared resources and services that large numbers of distributed clients wish to access, and for which we wish to control access or quality of service.*

- ***Problem:*** *By managing a set of shared resources and services, we can promote modifiability and reuse, by factoring out common services and having to modify these in a single location, or a small number of locations. We want to improve scalability and availability by centralizing the control of these resources and services, while distributing the resources themselves across multiple physical servers.*

- ***Solution:*** *Clients interact by requesting services of servers, which provide a set of services. Some components may act as both clients and servers. There may be one central server or multiple distributed ones.*



Client-Server Architecture Features

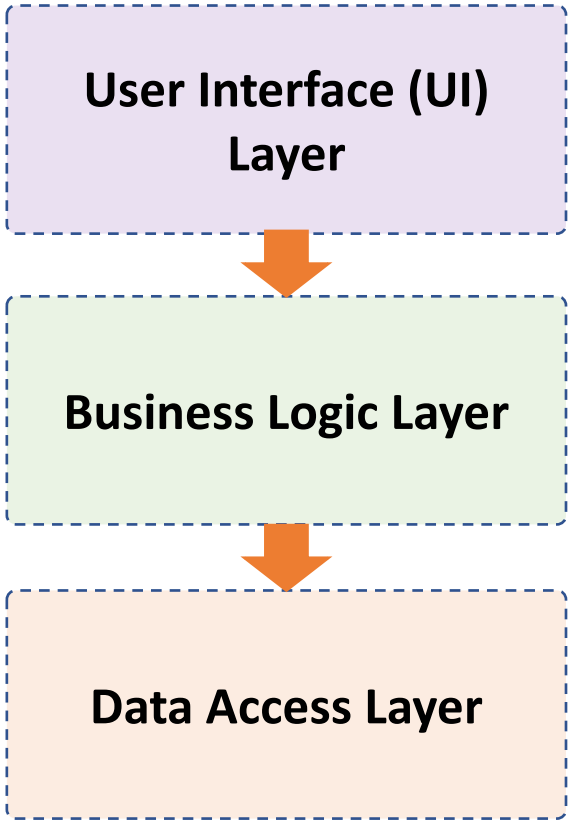
- Advantages
  - Servers can be distributed across a network.
  - General functionalities can be made available to all clients.
- Constraints
  - Clients are connected to servers through request/reply connectors.
  - A server component can also be a client to other servers.
- Weaknesses
  - Server might be a performance bottleneck.
  - Server might be a single point of failure.
  - Decisions about where to locate functionality (client or server) are often complex and costly to change after a system has been built.

Layered Architecture

- ***Context:*** *All complex systems experience the need to develop and evolve portions of the system independently. For this reason the developers of the system need a clear and well documented separation of concerns, so that modules of the system may be independently developed and maintained.*

- ***Problem:*** *The software needs to be segmented in such a way that the modules can be developed and evolved separately with little interaction among the parts, supporting portability, modifiability, and reuse.*

- ***Solution:*** *To achieve this separation of concerns, the layered pattern divides the software into units called layers. Each layer is a grouping of modules that offers a cohesive set of services. The usage must be unidirectional. Layers completely partition a set of software, and each partition is exposed through a public interface.*



Layered Architecture Features

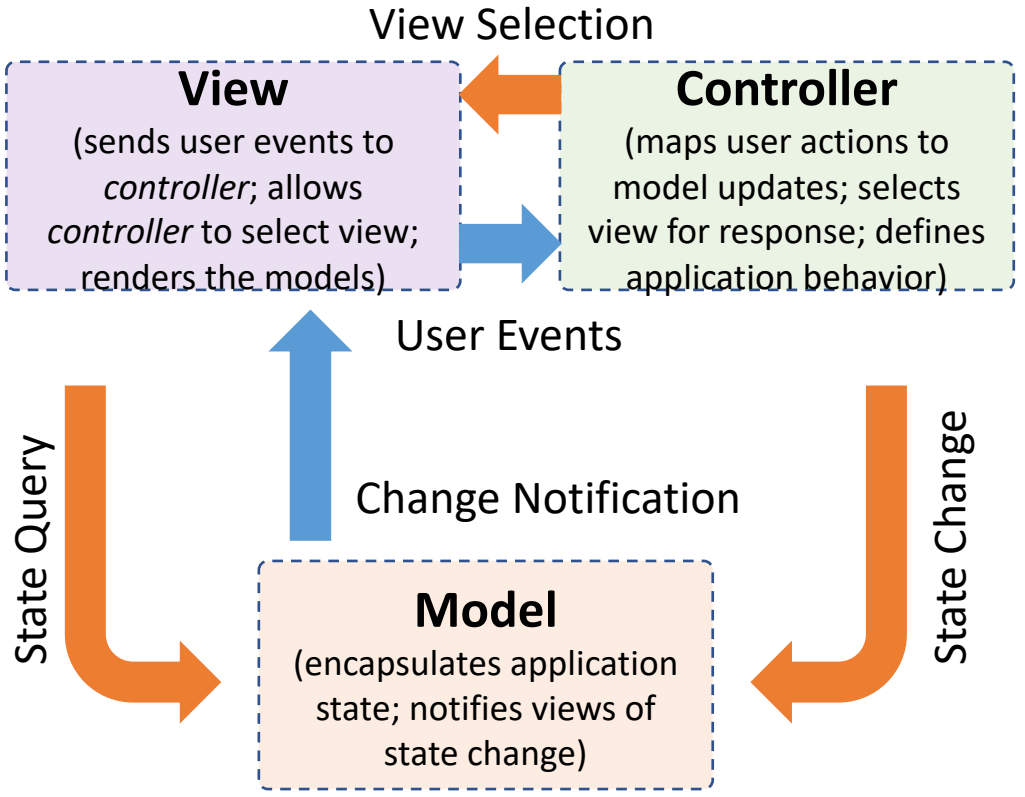
- Advantages
  - Allows replacement of entire layers if the interface is maintained.
  - Redundant facilities can be provided in each layer to increase the dependability of the system.
- Constraints
  - Every piece of software is allocated to exactly one layer.
  - There are at least two layers (but usually three or more layers).
  - The *allowed-to-use* relations should not be circular (i.e., a lower layer should NOT use a layer above).
- Weaknesses
  - The addition of layers adds up-front cost and complexity to a system.
  - Layers contribute a performance penalty.

Model-View-Controller (MVC) Architecture

- ***Context***: *User interface is typically the most frequently modified portion of an interactive application. Users often wish to look at data from different perspectives. These representations should both reflect the current state of the data.*

- ***Problem***: *How can user interface functionality be kept separate from application functionality and yet still be responsive to user input, or to changes in the underlying application's data? And how can multiple views of the user interface be created, maintained, and coordinated when the underlying application data changes?*

- ***Solution***: *The model-view-controller (MVC) pattern separates application functionality into three kinds of components:*
  - *A model, which contains the application's data*
  - *A view, which displays some portion of the underlying data and interacts with the user*
  - *A controller, which mediates between the model and the view and manages the notifications of state changes*



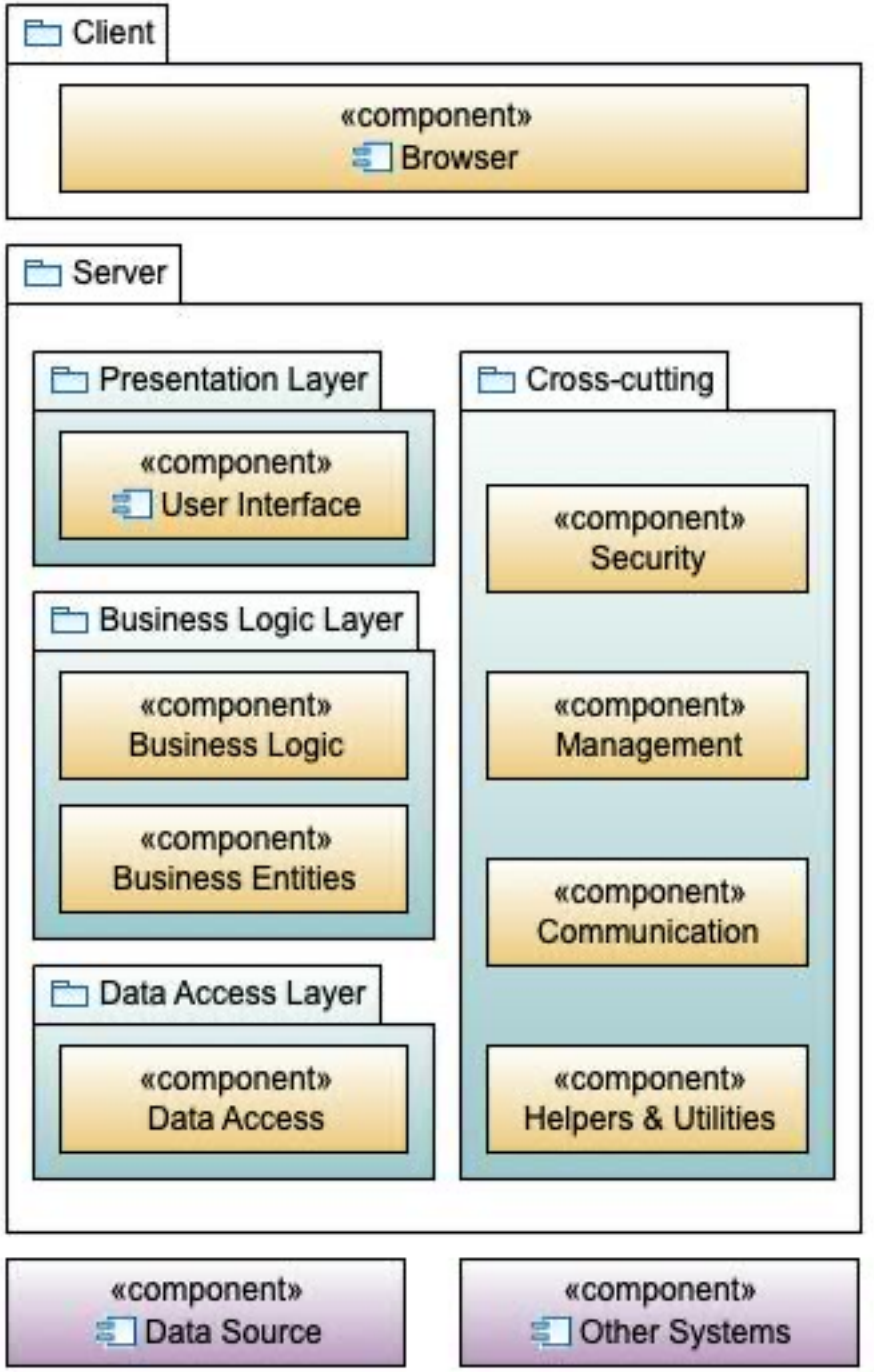
MVC Architecture Features

- Advantages
  - Allows the data to change independently of its representation and vice versa.
  - Supports presentation of the same data in different ways.
- Constraints
  - There must be at least one instance each of model, view, and controller.
  - The model component should not interact directly with the controller.
- Weaknesses
  - The complexity may not be worth it for simple user interfaces.
  - It may not be good fits for some user interface toolkits.

Reference Architectures – Web Application

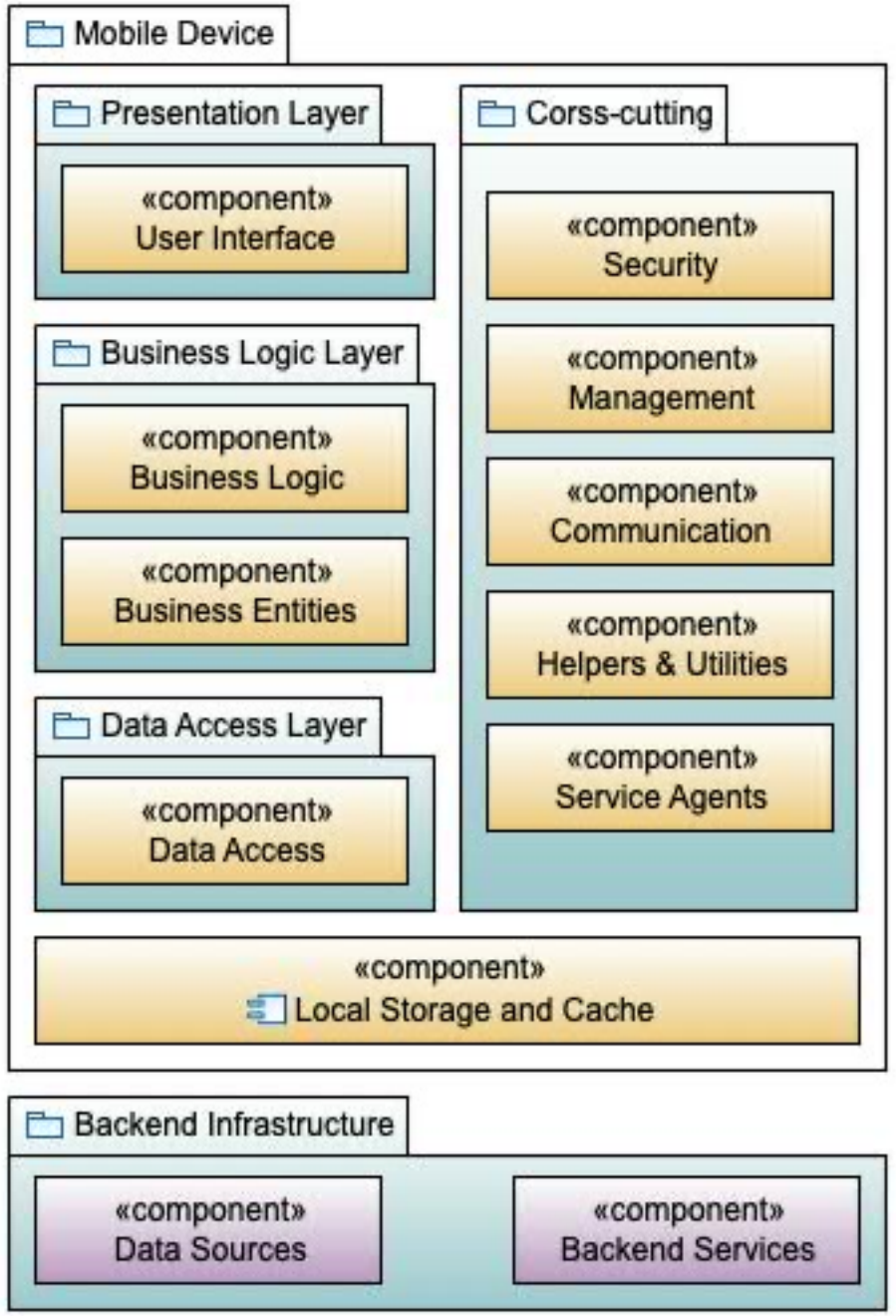
- Reference architectures are blueprints that provide an overall logical structure for particular types of application.
- A web application is typically accessed through web browsers that communicates with backend servers using HTTP protocol.
- Consider using the reference architecture when:
  - Don’t require a rich user interface
  - Don’t want to install any client software
  - Portability and accessibility are importation to the software architecture
  - Minimize the use of client-side resources

HTTP: Hypertext Transfer Protocol



Reference Architectures – Mobile Application

- A mobile application is typically installed on a mobile devices and supported by a backend infrastructure.
- Consider using the reference architecture when:
  - the mobile application has two parts. A client part that runs on a mobile device and a service part that runs on a backend server such as a cloud environment.
  - The network connectivity is unreliable. The client software may be run without the backend services when network is unstable.



Architectural Design Process '*Rules of Thumb*'

- Software architecture should be the product of a single architect or a small group of architects
  - Better conceptual integrity and technical consistency.
- The architect(s) should base the architecture on a prioritized list of well-defined quality attribute requirements.
- The architecture should be evaluated for its ability to deliver the system's important quality attributes.
- The architecture should NOT depend on a particular version of a commercial product or tool.

“Good Design” vs. “Bad Design”

“A good architecture is one that successfully address the concerns of its stakeholders and, when those concerns are in conflict, balances them in a way that is acceptable to the stakeholders.”

-- Rozanski, N. and Woods, E., 2012. Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley.

“There is no such thing as an inherently good or bad architecture. Architectures are either more or less fit for some purpose.”

-- Bass, L. Clements, P. and Kazman, R. Software architecture in practice, 3ed, 2013.