# SE202
# Introduction to Software Engineering

## Lecture 5-2
## Architecture patterns

Pathathai Na Lumpoon

# Last class

- What is design?
- High level design
  - What to specify
- Parts of a system
- Top-down versus bottom-up design
- Principles Leading to Good Design

# Today

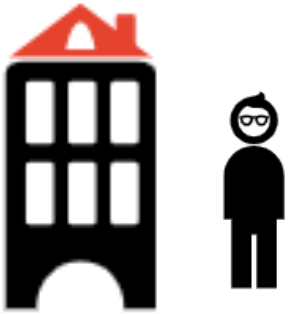
- Software architecture
- Architecture patterns

# Software Architecture (SWA)

- *Software architecture* is the process of designing the global organization of a software system, including dividing software into subsystems, deciding how these will interact, and determining their interfaces.
- Four main reasons to develop an architecture model
  - To enable everyone to better understand the system
  - To allow people to work on individual pieces of the system in isolation
  - To prepare for extension of the system
  - To facilitate reuse and reusability

# Prescriptive vs Descriptive Architecture

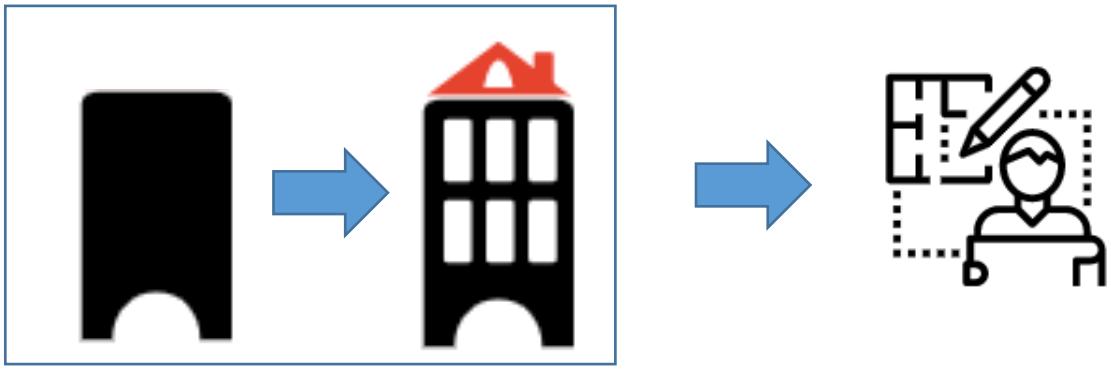
- A prescriptive architecture captures the design decisions make prior the system's construction
  - As conceived SWA



- A descriptive architecture describes how the system has actually been built
  - As implemented SWA

# Architectural evolution

- When a system evolves, ideally its prescriptive architecture should be modified first



- In practice, this rarely happens
  - Developer's sloppiness
  - Short deadlines
  - Lack of documented prescriptive architecture
  - …

# Architectural degradation

- Architectural drift
  - Introduction of architectural design decision to a system's prescriptive architecture, but do not conflict with it

- Architectural erosion
  - Introduction of architectural design decisions that violate a system's prescriptive architecture

# Architectural Recovery

- Drift and erosion --> degraded architecture
- Two solution
  - Keep solving the code
  - Architectural recovery
    - Determine SWA from implementation and fix it

# How to develop an architecture model

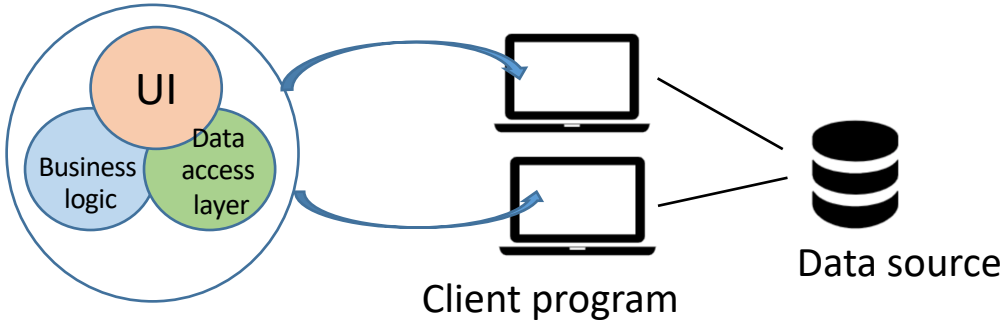
1. Start by sketching an outline of the architecture, based on the principal requirements, including the domain model and use cases.
2. Refine the architecture by identifying the main ways in which the components will interact, and by identifying the interfaces among them.
3. Consider each use case, adjusting the architecture to make it realizable.
4. Mature the architecture as you define the final class diagrams and interaction diagrams.

# Common architecture styles

- Monolithic
- Client/Server
- Multi-Layer architectural pattern
- Model–View–Controller (MVC) architectural
- Serverless Architecture
- Event-Driven Architecture
- Microservices Architecture

# Monolithic

- A single executable that performs all functions for an application.

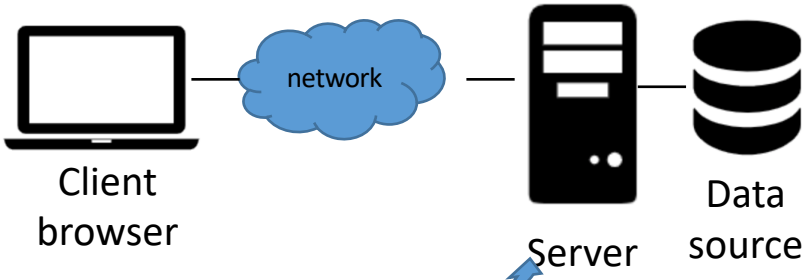



- The pieces of the system are tied closely together, so it doesn't give you a lot of flexibility.
- Good for small applications where a single programmer or team is working on the code.

# Client/Server

- A *client/server architecture* separates pieces of the system that need to use a particular function (clients) from parts of the system that provide those functions (servers).
- That decouples the client and server pieces of the system so that developers can work on them separately.

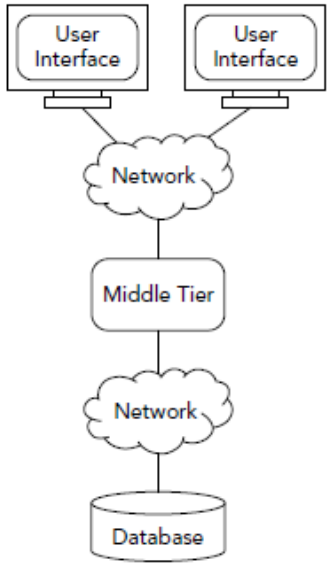
2-tier architecture




Monolithic approach

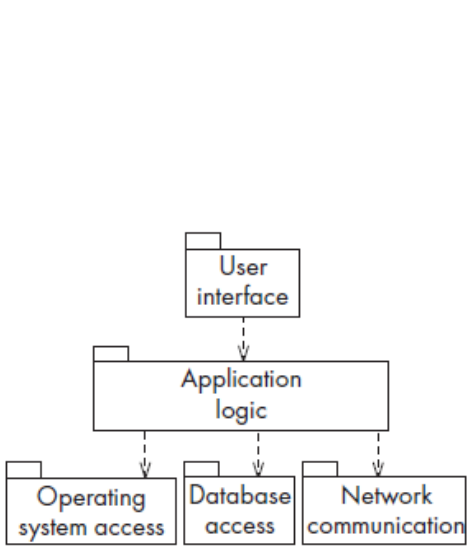
If a single executable that performs all of the server-side functions for an application.

3-tier architecture

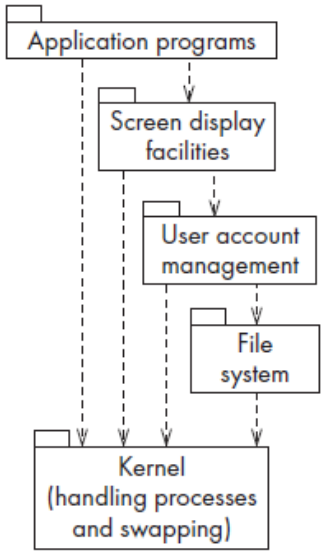


# The Multi-Layer architectural pattern

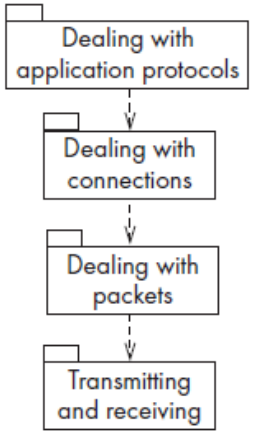
- This pattern can be used to structure programs that can be decomposed into groups of subtasks, each of which is at a particular level of abstraction. Each layer provides services to the next higher layer.




(a) Typical layers in an application program




(b) Typical layers in an operating system

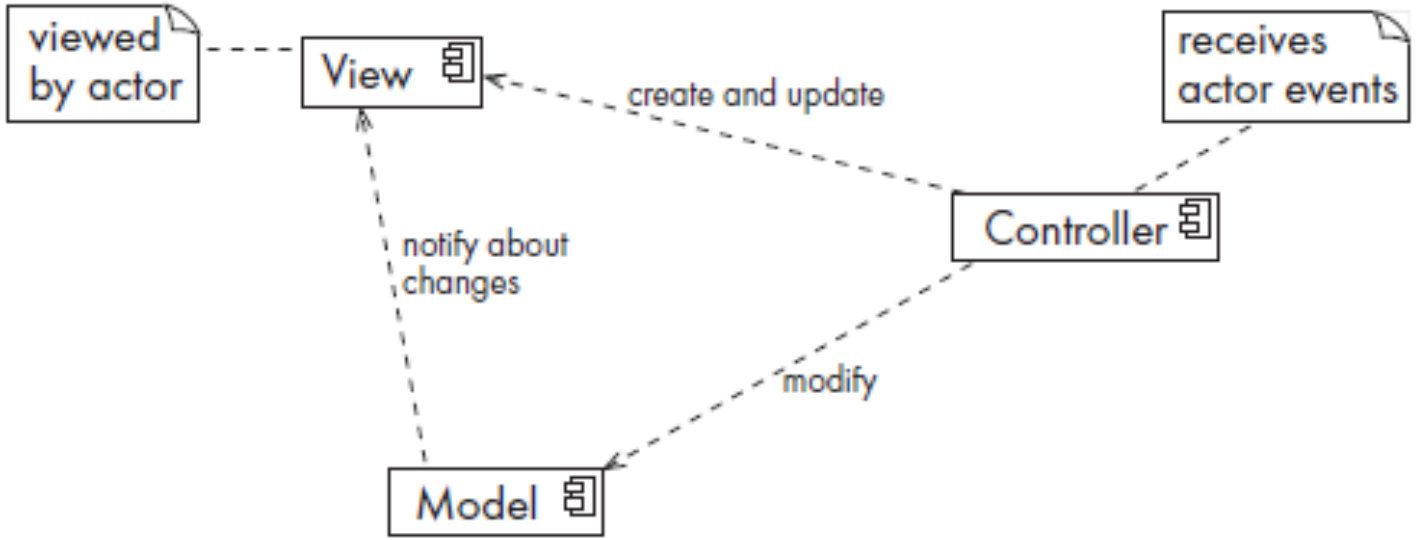



(c) Simplified view of layers in a communication system

# Model–View–Controller (MVC) architectural

- Model–View–Controller, or MVC, is an architectural pattern used to help separate the user interface layer from other parts of the system.
- The ***model*** contains the underlying classes whose instances are to be viewed and manipulated.
- The ***view*** contains objects used to render the appearance of the data from the model in the user interface. The view also displays the various controls with which the user can interact.
- The ***controller*** contains the objects that control and handle the user's interaction with the view and the model. It has the logic that responds when the user types into a field or clicks the mouse on a control.

# The Model–View–Controller (MVC) architectural pattern for user interfaces



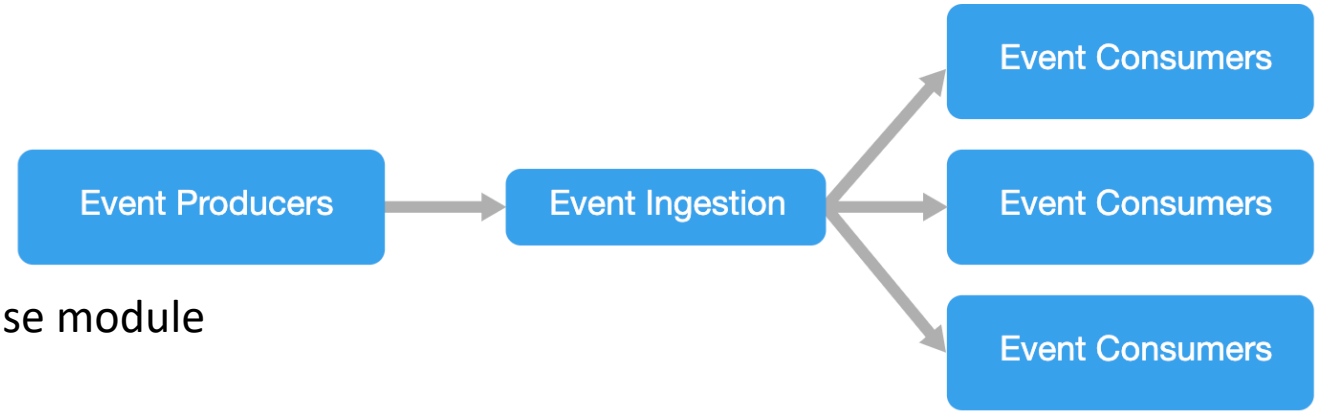
# Serverless Architecture

- Serverless Architecture depends on third-party services (on the cloud) to manage the complexity of the servers and backend management.
- 2 types
  - Backend as a service (BaaS)
  - Functions as a Service (FaaS)
- Save a lot of time taking care and fixing bugs of deployment and servers regular tasks
- E.g

# Event-Driven Architecture

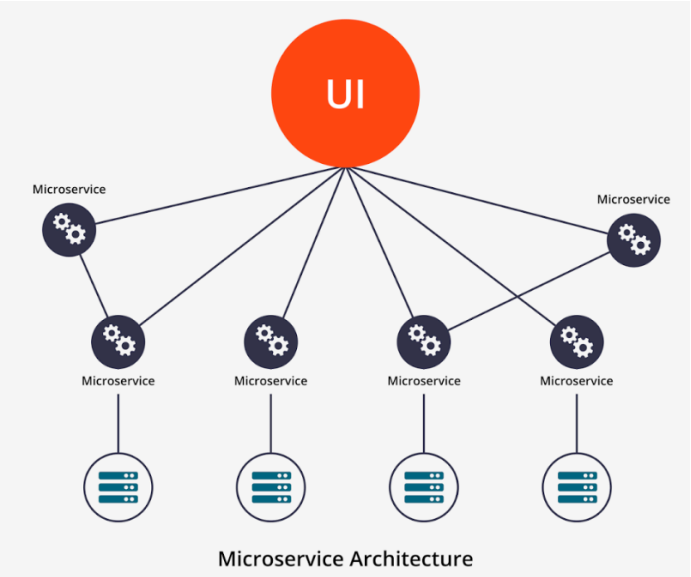
- Event-Driven Architecture depends on Event Producers and Event Consumers.
- The main idea is to decouple your system's parts and each part will be triggered when an interesting event from another part has got triggered.

A vendor module




A purchase module

# Microservices Architecture



Microservice Architecture

- Microservices architecture has become the most popular architecture in the last few years.
- It depends on developing small, independent modular services where each service solves a specific problem or performs a unique task and these modules communicate with each other through well-defined API to serve the business goal.
- The microservice architectural pattern is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.