# Chapter 5

## High Level Design

-To specify

-type of design

-principles of good Design

-Architecture patterns

- UML DB UI Design

# High level design

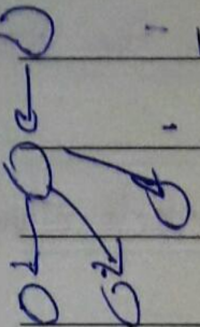
- view of the system at an abstract level
  - Show how major pieces of finished app fit together
  - Specifies the enviroment in Finish app will run
- allow different teams to work on them simultaneously
- not focus on the details of how the pieces will work

## What to specify

- Security : Network scurity, Physical security
- Operating system : Windows, ios, linux
- Hardware platform : desktop, laptop, tablet, phone, mainframe
- other hardware : networks, printer, papers, audio, video
- User interface style : navigational techniques, menus, screen, forms
- Internal interface : interaction between program piece
- External interface : interactiont with external system
- Algorithm : design of computational mechanisms
- Architecture : monolithic, client-sever, multitier, etc.
- Reports : application usage, customer purchase, work schedules
- Database : DB platform, major table, their relationships
- Top-level classes : Customer, Employee, Order
- Data flows : Flow of data among different process

# Top down Design

- Design very high level structure
- work down to detailed decision about low level
- detailed decision
    - format particular data items
    - individual algorithms that will be used

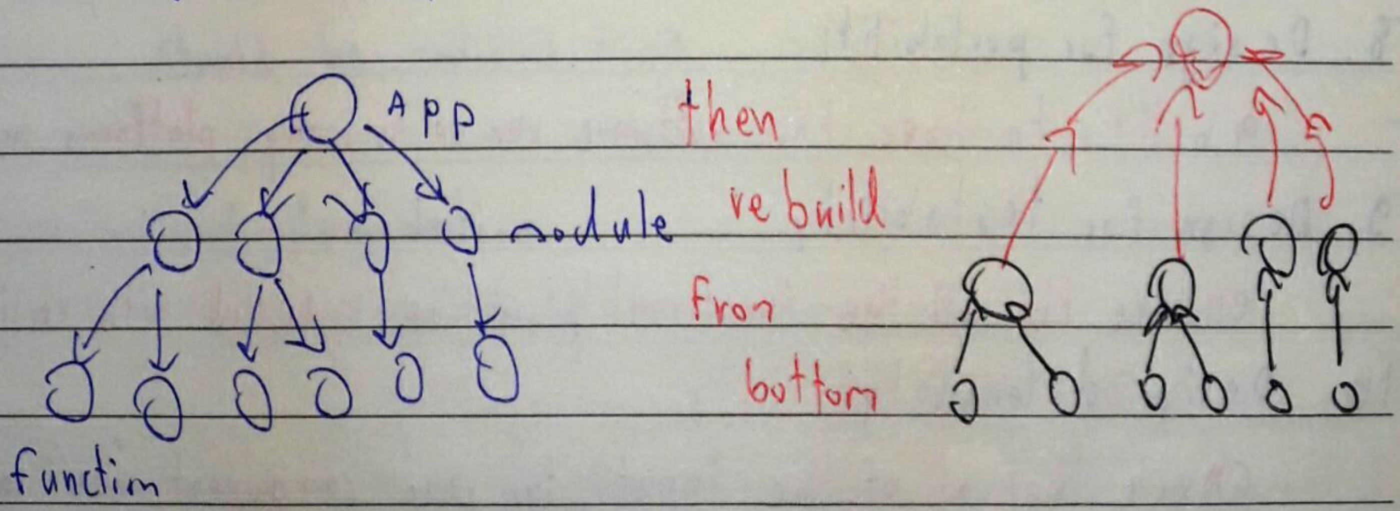


# Botton-up Design

- Make decisions about reusable low-level utilities
- decide how these will be put together to create high level constructs

"A mix of top down and bottom-up approaches are normally used"

Design Whole system first



goals of good design

- usability
- Efficiency
- Reliability
- maintainability
- Reusability

## Principles Leading to Good Design

1. Divide and Conquer

Dividing things up into smaller chunk to achieve the goal

2. Increase cohesion where possible

An entity keep togethor things that are related to each other

3. reduce coupling where possible

reduce the number of connection between modules

4. keep the level of abstraction as high as possible

understand the essence of something and make decisions without unneecessary detail

5. Increase reusability where possible

Generalize a design as much as possible

6. Design for flexibility

anticipating changes that design may have to undergon in the future

7. Anticipate obsolescence

planning For evolution of the technology or environment

8. Design for portability

ability to have the software run on as many platforms as possible

9. Design for testability

ensure that all the functionality can be executed with various input

10. Design defensively

check that all of the inputs to your component are valid.

## Techniques for making good design decisions

- Using objectives and priorities

"The qualities to consider when setting priorities and objective include memory efficiency, CPU, maintain, portability, usability

# Architecture patterns

## Software Architecture (SWA)

- process of designing the global organization
- Four reasons to develop an architecture model
  - to enable every one to better understand the system
  - allow people to work on idividual pieces of the system
  - prepare for extension of the system
  - facilitate reuse and reusability

| (conceived SWA) | | (Implemented SWA) |
|---|---|---|
| Perscriptive | VS | Descriptive |
| - capture the design Decision<br>- Make prior the system construction | | - Describe how System has actually been built |

## Architectural evolution

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

## Architectural degradation

- Architectural drift
  - System prescriptive but don't conflict with it
- Architectural erosion
  - violate a system's prescriptive

Architectural Recovery

- Drift and Erosion → degraded architecture
- keep solving the code
- Determine SWA from implementation and fix it

How to develop an architecture model

1 sketching an outline of architecture
  - priciple requirement
  - domain model
  - Use case

2 Refine the architecture
  - identifying the main way
  - which component will interact
  - identifying the interface among them

3. Consider
  - each use case
  - adjusting the architecture to make it realizable

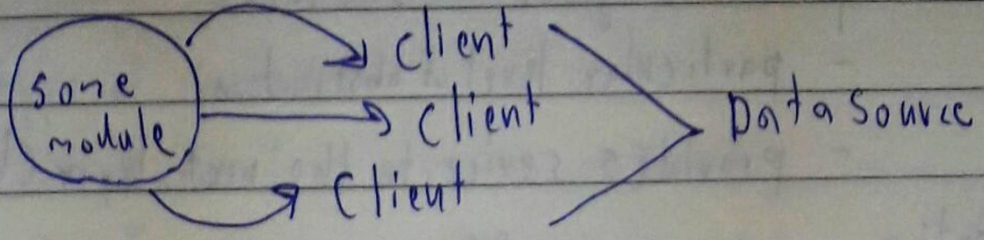
4. Mature the architecture as you define the final class diagrams and interaction diagrams

Common architecture styles

- Monolithic
- Model-View-Controller (MVC)
- Client/server
- Severless
- Even-Driven
- Multi-Layer architectural pattern
- Microservices

## Monolithic

- Single executable that performs all function for an application

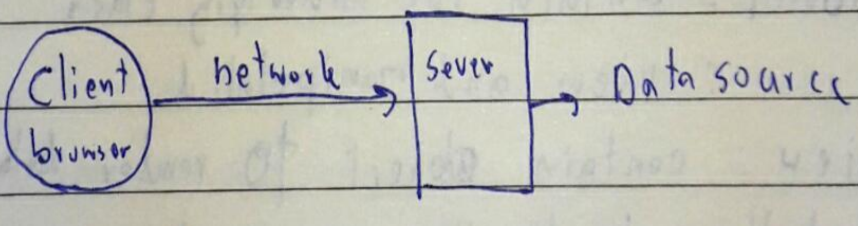


- not so flexability
- good for small application

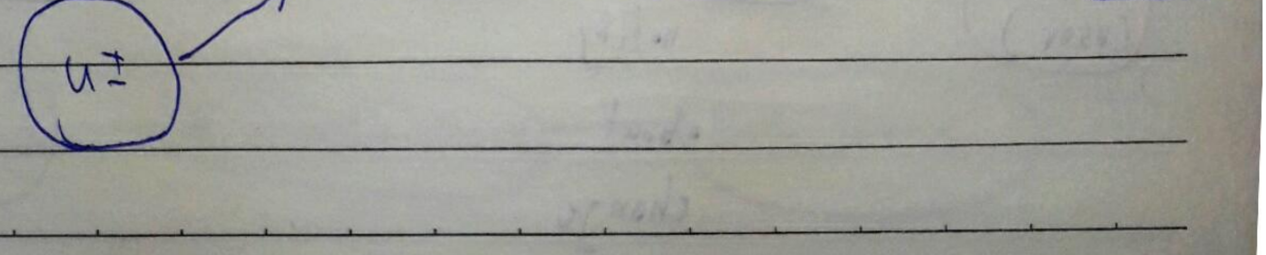
## Client / Server

- Separates piecs of the system that need a particular function from parts of the system that provide those function
- That decouples the client and server piece of the system so that developers can work on them separately

2 tier

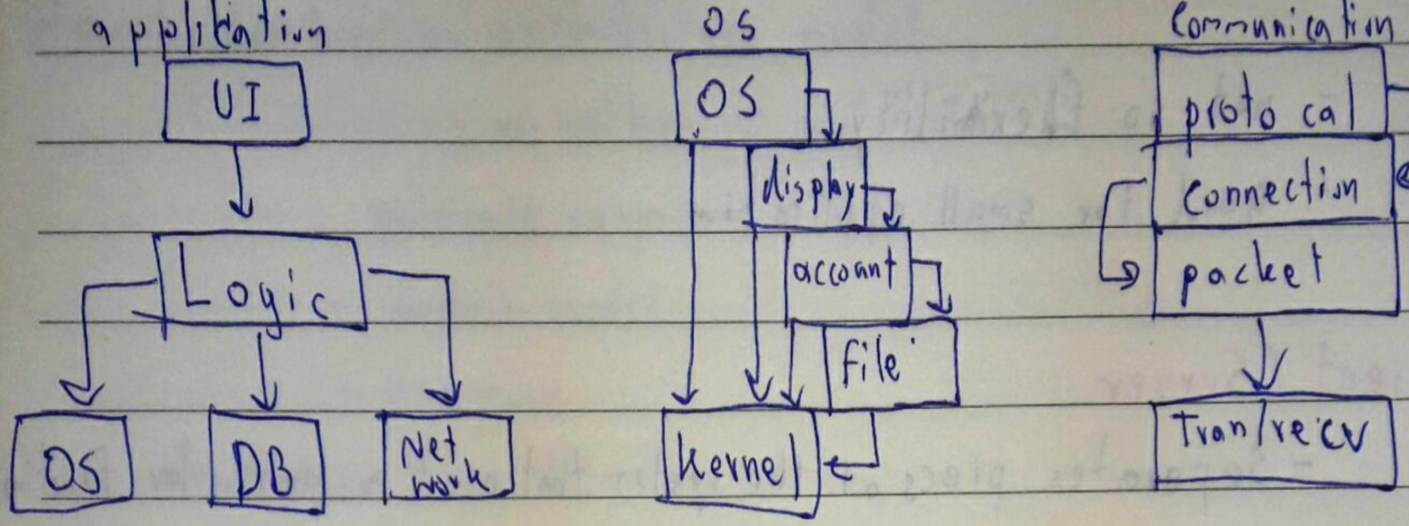


3 tier



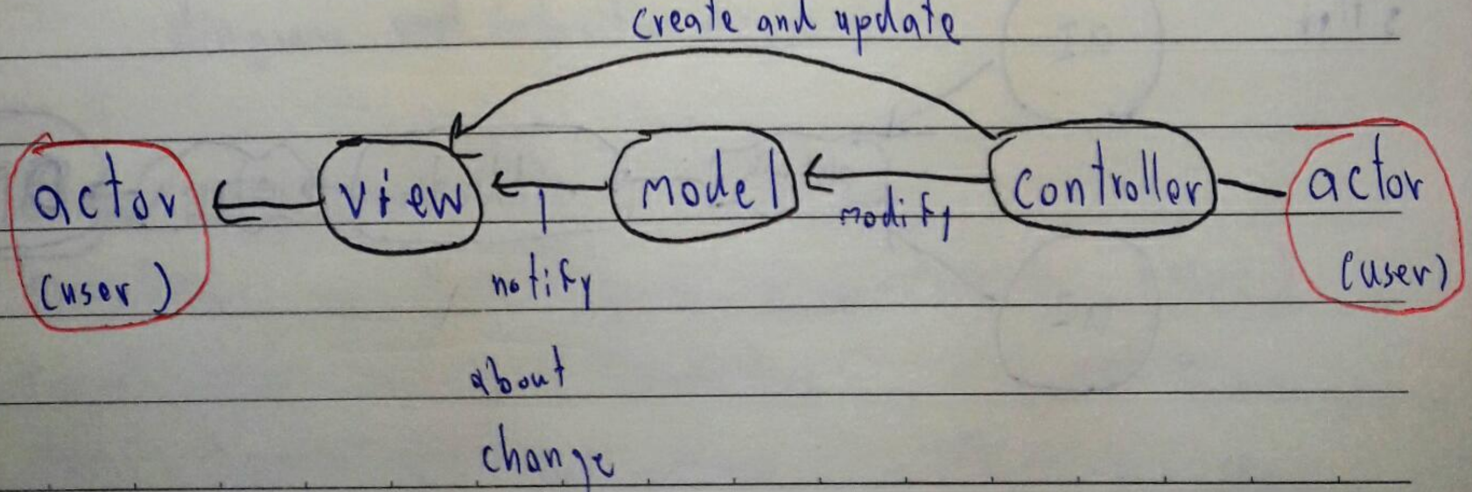
# The multi-layer architectural pattern

- This pattern can be used to structure programs that can decomposed intogroup of sub tasks,
  - particular level of abstraction
  - provides sevice to the next layer (higher)



## Model-View-Controller (MVC)

- separate User interface layer from other part of the sys
- Model : contain the underlying class
  : view and manipalated
- View : contain object to render data from UI(model)
- Controller : handle the user Interaction with view and model

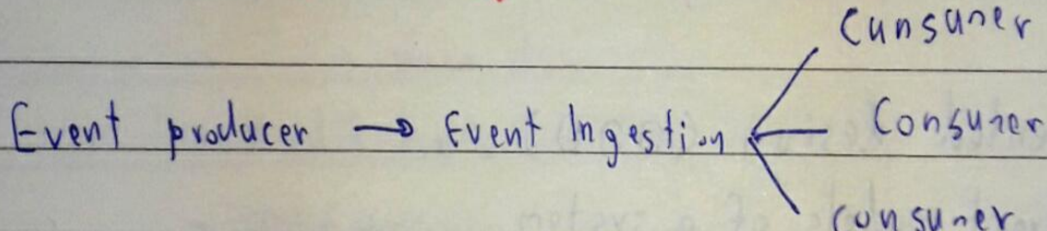


## Serverless Architecture

- depends on third-party services ( cloud )
  to mange the complexity of servers and backend
- 2 type of Serverless
  - Backend as a service (BaaS)
  - function as a service (FaaS)
- Save a lot of time taking care and fixing bug

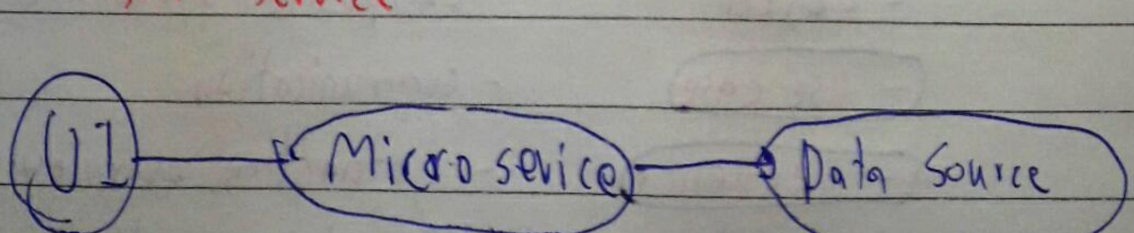
## Event-Driven Architecture

- main idea is to decouple system part
  - each part will be triggered when interesting event got triggered



## Micro service

- most popular architecture in the last few year (2017 - 2020) (now)
- depend on small independent modular service, API,
- approach to developing a single application as suit of small service



UML Data base design User Interface design

Different aspects of design

- architecture design:
  - The division in to subsysten and components
    - How it gonna connect
    - How it interact
    - Interface
- User interface design
- Data base design
- Class design: the various of features
- Algorithm design: the computational mechanisms
- Protocol design: communications protocol.

Object-oriented design (OOD)

- abstract models of a system
- Represent the models by graphical notation
- Mostly use Unified Modeling language (UML)
  - timing
  - class
  - activity
  - object
  - use case
  - sequence
  - package
  - state
  - component
  - composite structure
  - communication
  - interaction overview
  - deployment

Common Diagron

- Different models present different perspectives

## Database design

- database is an organized collection of related information
- Data in database
    - records, files, object
    - independent from applications using it
    - have entities and relation ship among then
    - entity is a thing, person, object, any iten that should be captured and stored in the table

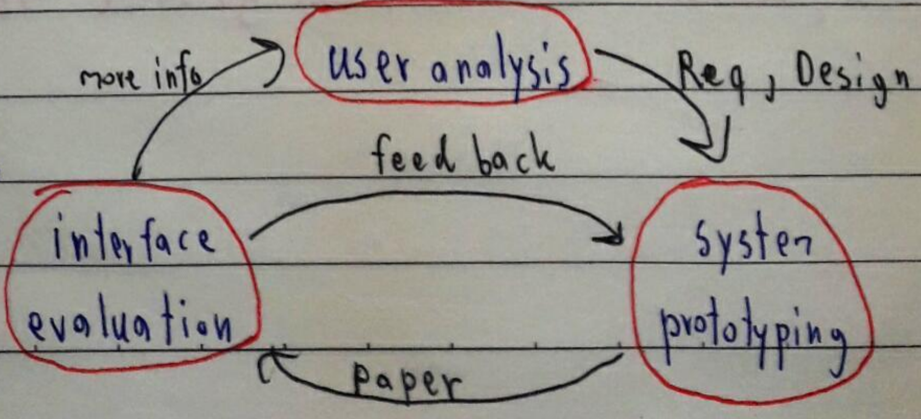
Design the database

- kind of database the program will need
    - Text, files, object store
    - Table (relational database)
    - data structure
- Audit trails (history table)
- User access

## USER interface design

- iterative process involving close chanels between user and designers

three core activities



## Wireframes

- layout of web page that demonstrates what interface elements will exist on key pages
- it is a critical part of the interaction design process
- to provide a visual understanding of page in early project

We can use

- paper, pen, pencil
- Power point
- Balsamiq
- Adobe XD

## User Interface Design Principles

User Familarity: The terms and concepts are familiar to the user

Consistency: operations should be activated in the same way.

Minimal Surprise: the user should be able to predict the operation

Feed back: Provide the user with visual and auditory feed back

Memory load: Minimize the memory load.

Efficiency: Minimize keystorkes and mouse movements

Recoverability: recover from their error, undo facilities

User guidance: context- sensitive, user guidance assitance