

SOFTWARE ARCHITECTURES INTRODUCTION

Software Engineering Seminar

Author: Eng. Carlos Andrés Sierra, M.Sc.
cavirguezs@udistrital.edu.co

Full-time Adjunct Professor
Computer Engineering Program
School of Engineering
Universidad Distrital Francisco José de Caldas

2025-III



Outline

1 Systems Thinking



2 Reference Architectures



Outline

1 Systems Thinking

2 Reference Architectures



What is a System?

- A **system** is a set of interacting components that work together to achieve a common goal.
- A system is a collection of elements that are organized in a specific way. → **modules - classes**
- A system is a structure that is designed to perform a specific function. → **architecture**

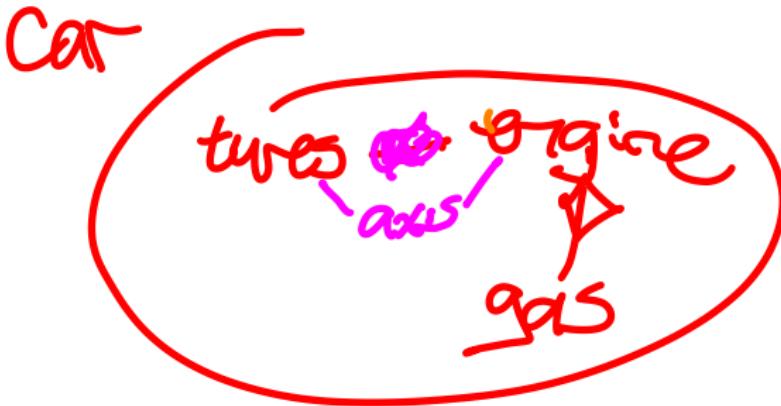
A - B - C

.



Systems Analysis Process

- **Systems analysis** is the process of studying a system in order to identify its components, interactions, and goals.
- **Systems analysis** is the process of understanding how a system works and how it can be improved.
- **Systems analysis** is the first step in the systems development lifecycle.



Systems Analysis Techniques

• **Systems analysis** uses a variety of techniques to study a system.

- It includes interviews, surveys, observations, and document analysis.
- It also includes data modeling, process modeling, and requirements analysis.

final user

stakeholders

business processes

flowchart

diagrams

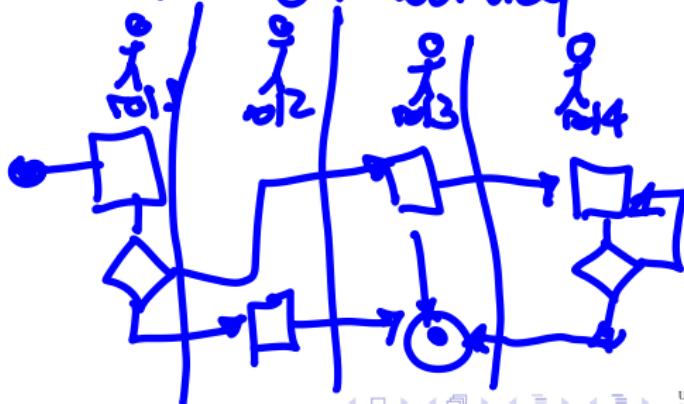


Systems Analysis Techniques

- Systems analysis uses a variety of techniques to study a system.
- It includes interviews, surveys, observations, and document analysis.
- It also includes data modeling, process modeling, and requirements analysis.

- Sources
- Format + structure
- Understanding

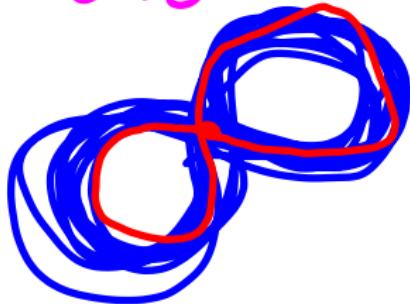
BPMN
sequence + activity



Complexity and Emergence

- Complexity is the *degree* to which a system is **difficult** to understand.
- Emergence is the *appearance* of unexpected properties in a system through interactions of its components.
- Complexity and emergence are common in many systems that are non-linear and stochastic.
- They can be *studied* and understood through systems analysis and modeling.

Lorenz



chaos theory

Randomness
(stochastic)

non-linear

chaotic
attractors



Complexity and Emergence

error fontime

- Complexity is the degree to which a system is difficult to understand.
- Emergence is the appearance of unexpected properties in a system that arise from the interactions of its components.
- Complexity and emergence often occur in dynamic systems that are nonlinear and chaotic.

swarm intelligence



Complexity and Emergence

- **Complexity** is the *degree* to which a **system** is **difficult** to understand.
- **Emergence** is the *appearance* of **unexpected properties** in a **system** that **arise** from the **interactions** of its **components**.
- **Complexity and emergence** are common in **dynamic systems** that are **non-linear** and **chaotic**.
- They can be *studied* and understood through **systems analysis** and **modeling**.

Complexity → elements + relations

Emergence → rules + random rules

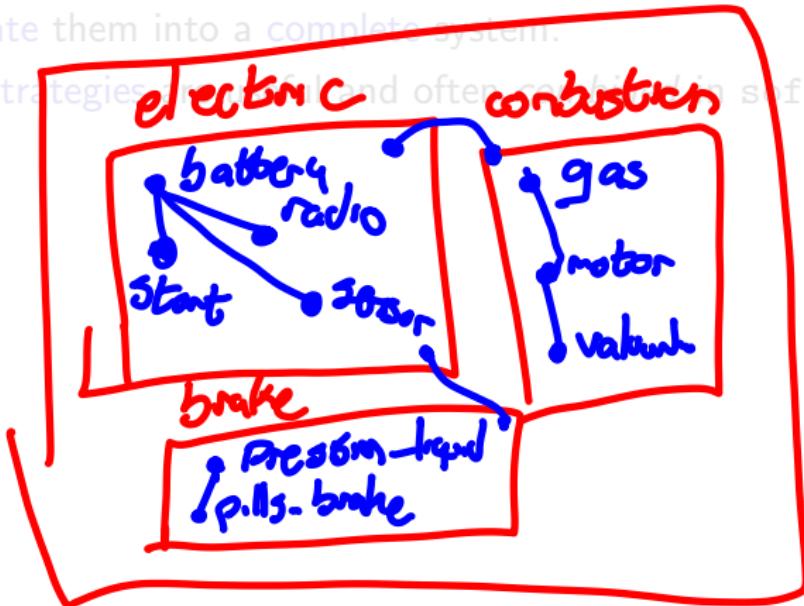


Strategies to Solve Problems

- **Top-Down:** Start from the big picture and break it down into smaller parts.
C.O.P.
 - **Bottom-Up:** Start from small, well-defined *car* components and integrate them into a complete system.
 - Both strategies *electronic combustion* and often used in software design.

C.O.P.

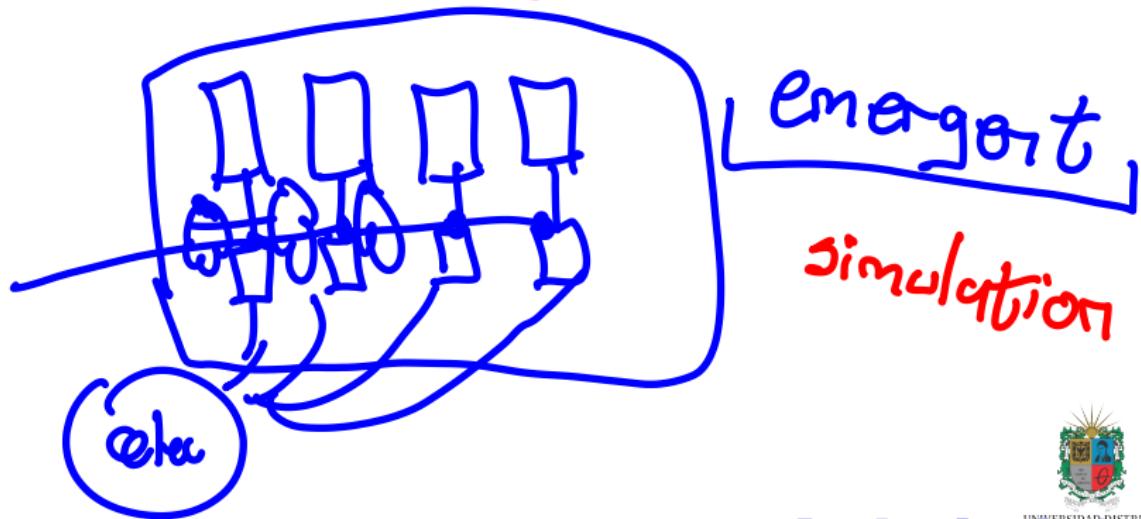
Сар



Strategies to Solve Problems

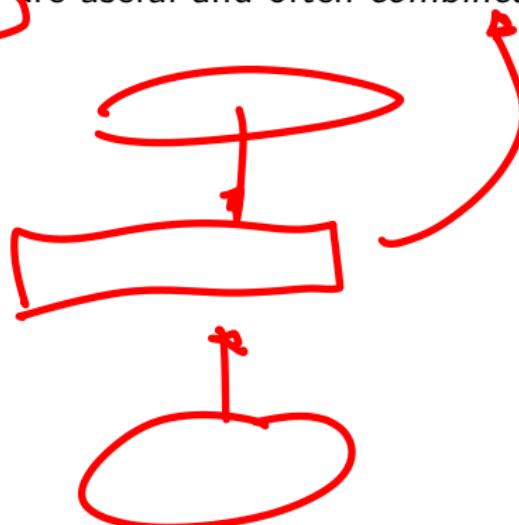
- **Top-Down:** Start from the **big picture** and **break** it down into smaller parts.
- **Bottom-Up:** Start from small, well-defined components and **integrate** them into a **complete system**.

↳ more understanding / more difficult



Strategies to Solve Problems

- **Top-Down:** Start from the **big picture** and **break** it down into smaller parts. → **Software**
- **Bottom-Up:** Start from **small**, well-defined components and **integrate** them into a **complete system**.
- **Both strategies** are useful and often *combined* in software design.



Process Definition

- A **Process** is a **series** of steps or actions taken to achieve a particular end.
- Processes are used to **organize** and **manage** work.

algorithm → code

business/

no tech

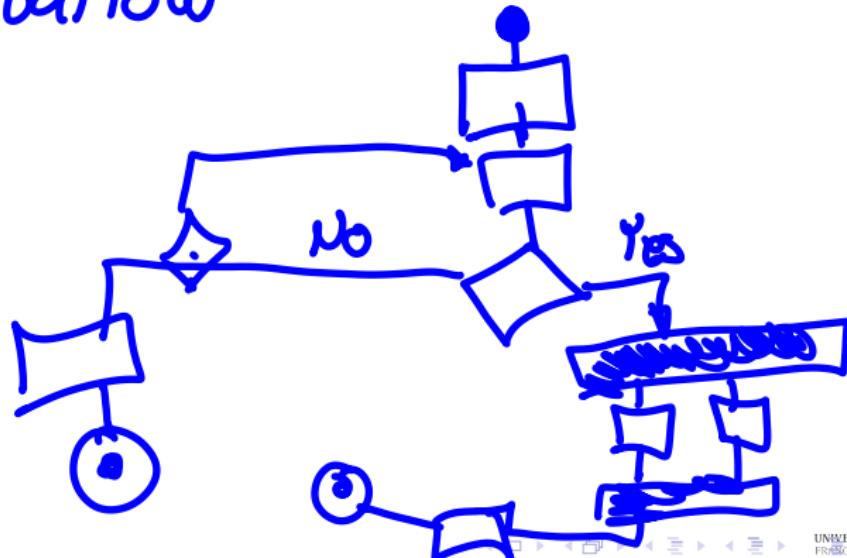


solution
software



Workflows→ business process

- A Workflow is a series of tasks that are performed in a specific order to achieve a goal.
- Workflows are used to automate and optimize business processes.
- Workflows can be sequential, parallel, conditional, or repetitive.

ETL - Dataflow

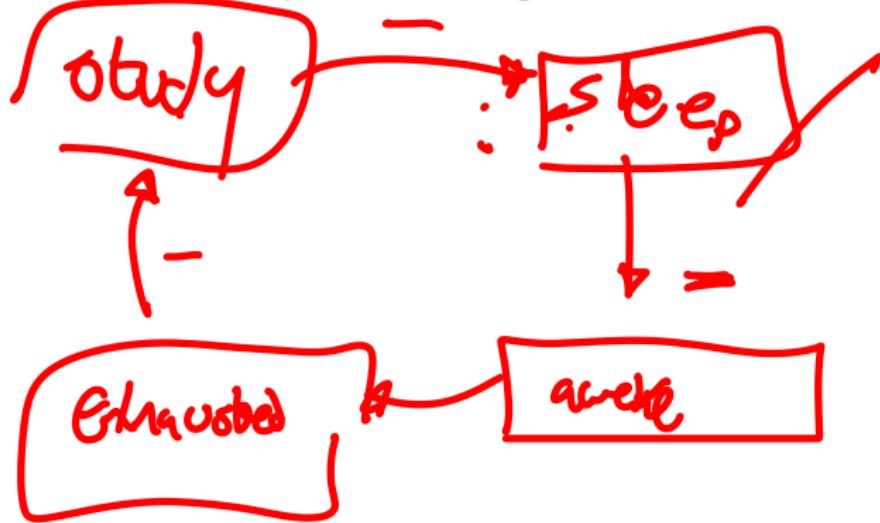
Process Models

- A **Process Model** is a representation of a process that shows the sequence of steps and the **relationships** between them.
- **Process models** are used to **analyze**, **design**, and **improve** processes.
- Examples of process models include **flowcharts**, **data flow diagrams**, **activity diagrams**, business process model and notation (BPMN), petri nets, state diagrams, among others.

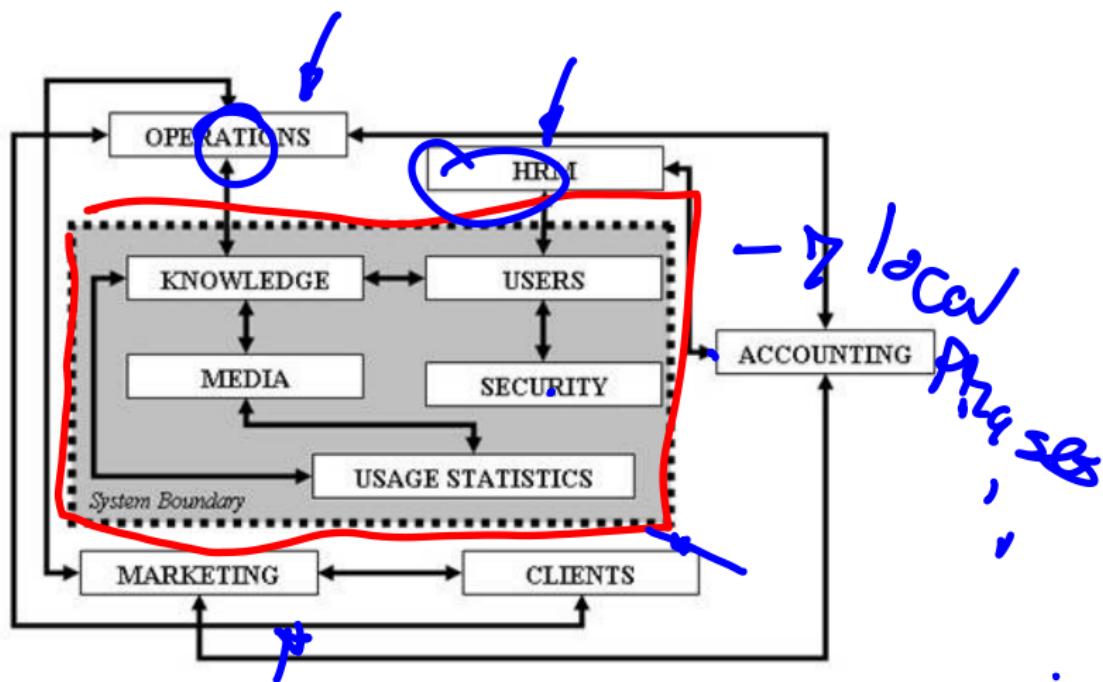


Causal Loops

- A **Causal Loop** is a diagram that shows the relationships between different variables in a system.
- Causal loops are used to analyze and understand the dynamics of a system.
- Causal loops can be positive or negative.

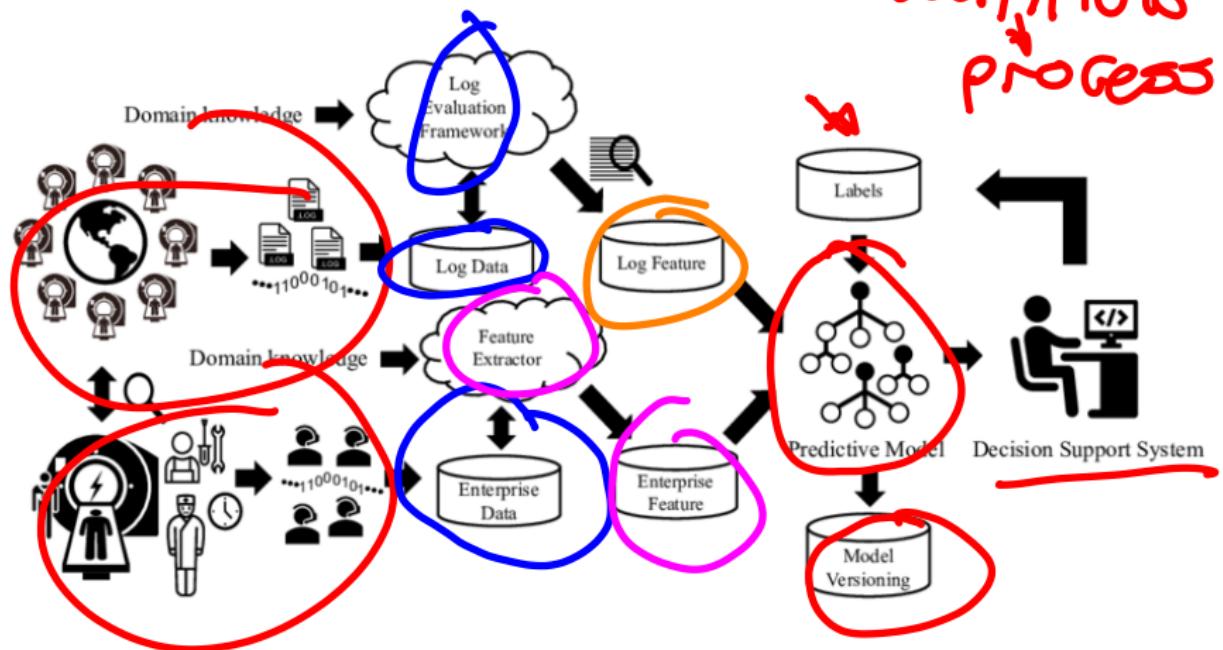


System Schema Example: Company Structure

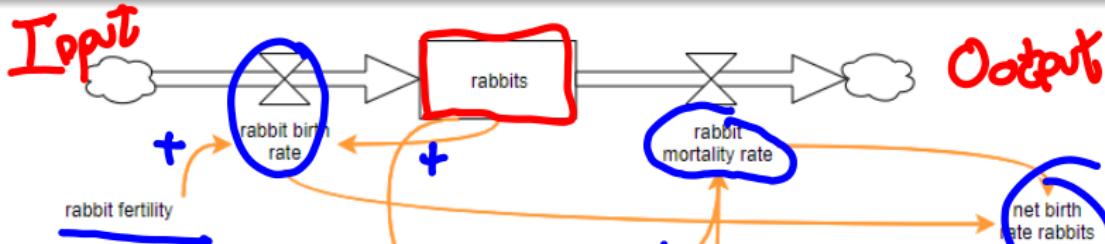


System Schema Example: Processing Pipeline

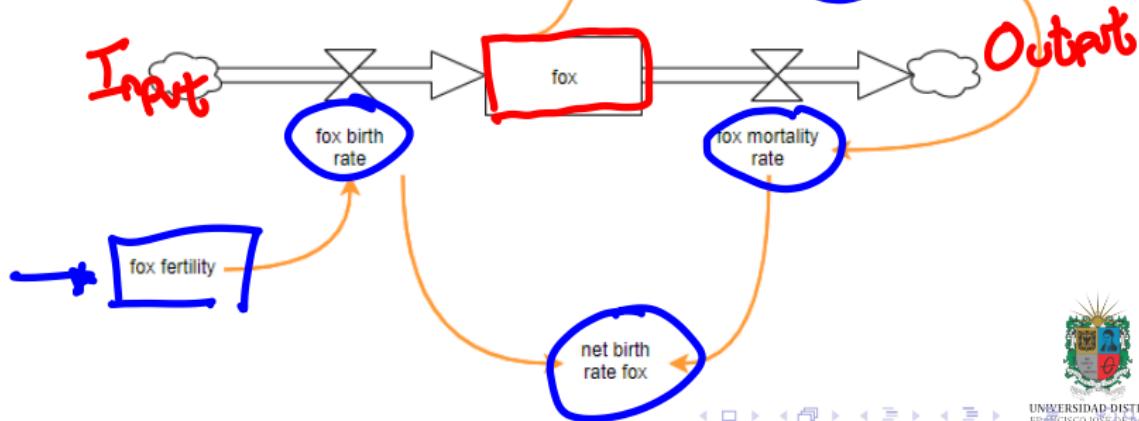
Workflow + process



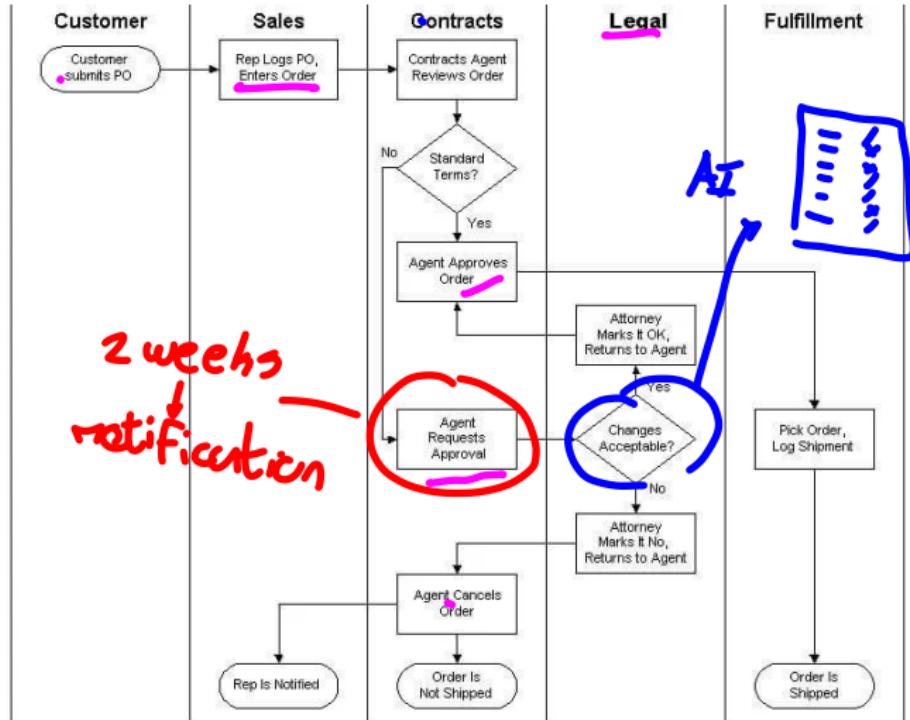
Stock and Flow Diagram



Stella



Business Process Model and Notation (BPMN)



Outline

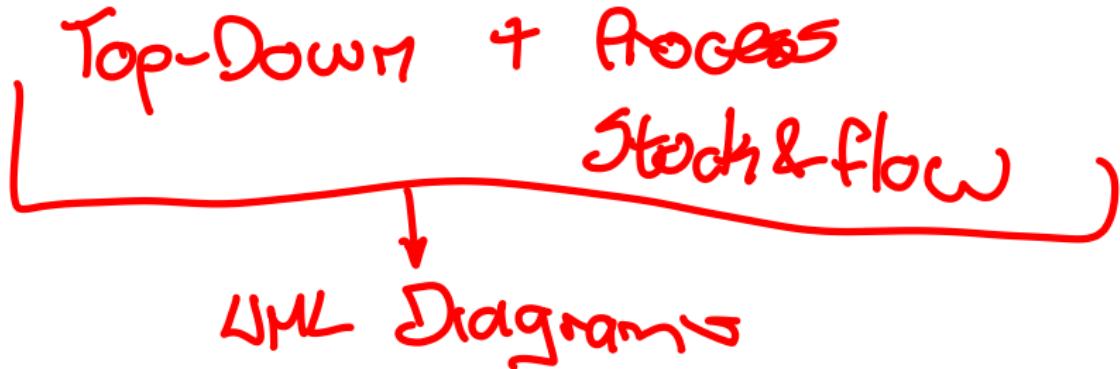
1 Systems Thinking

2 Reference Architectures



Design Before Code

- Design should come before coding.
- Jumping into code without a plan leads to confusion and rework.
- Good design clarifies the problem and guides the solution.



Understanding the Requirements

- Requirements must be well understood before design.
- Ask questions to clarify ambiguities and document all requirements.
- Requirements define the scope and direction of the design.

Implications - Trade-offs



Design Based on the Problem

- Design should be driven by the problem, not by technology.
- Focus on what needs to be solved, not just how to implement it.
- Use the **problem statement** to identify key objects and their relationships.

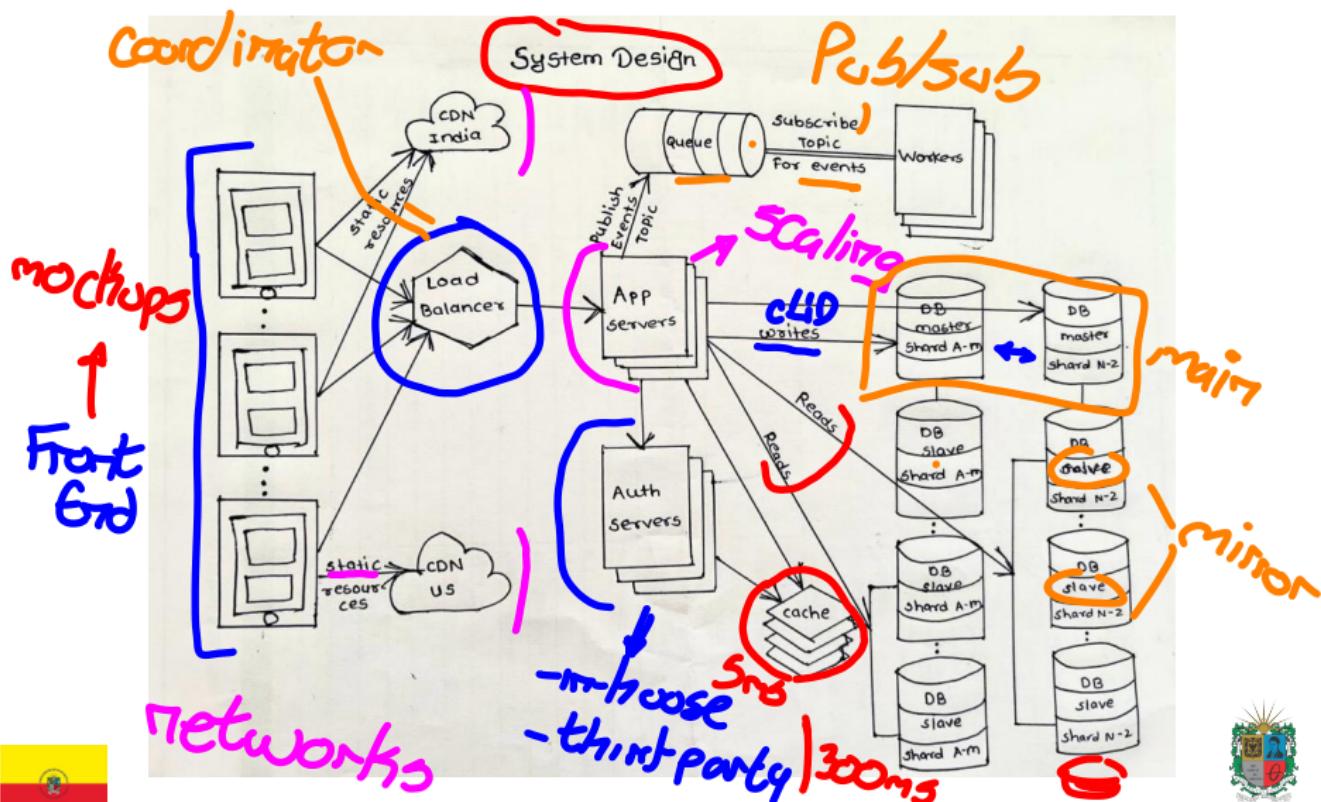
business

system

identify
problems



Systems Design applied to Software Architectures



Conceptual Design and Technical Design

- **Conceptual Design:** What the system should do, using high-level models.

- Technical Design: How the system will be implemented, using detailed diagrams and specifications.

- Both are essential for a successful software project.

understand goals + scope
+
all stakeholders

- Mockups
- CRC cards
- Business Process
BPMN



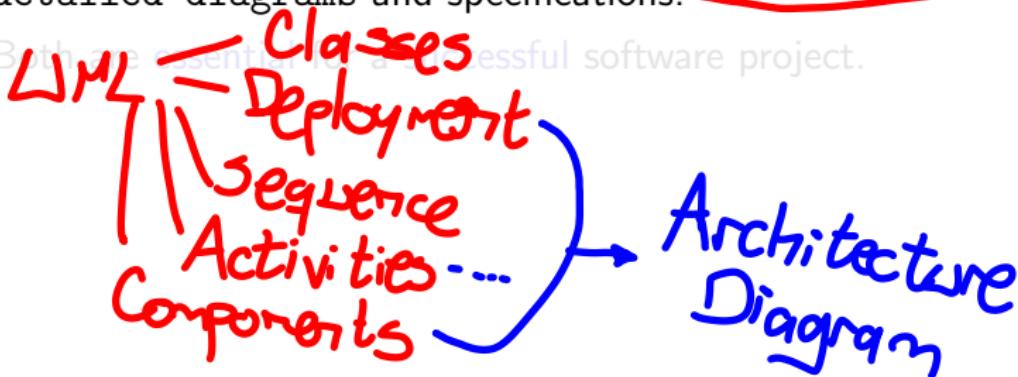
Conceptual Design and Technical Design

- **Conceptual Design:** What the system should do, using high-level models.

Negotiation

- **Technical Design:** How the system will be implemented, using detailed diagrams and specifications.

- Both are essential for a successful software project.



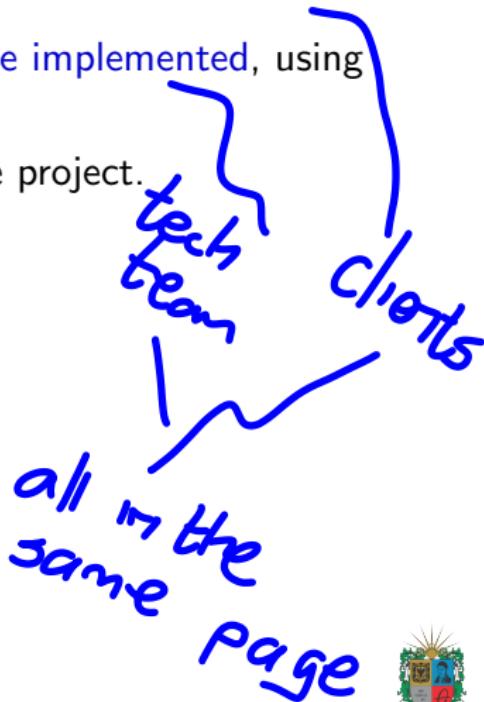
- DataBase System Design



Conceptual Design and Technical Design

- **Conceptual Design:** What the system should do, using high-level models.
- **Technical Design:** How the system will be implemented, using detailed diagrams and specifications.
- Both are essential for a successful software project.

Scope
Implications



What is a System Architecture?

- A **system architecture** is the structure of a system that defines its components, interactions, and relationships.
- A **system architecture** is the blueprint of a system that guides its development and implementation.
- A **system architecture** is the foundation of a system that ensures that it meets the needs of its users.

solve a
problem



Types of System Architectures

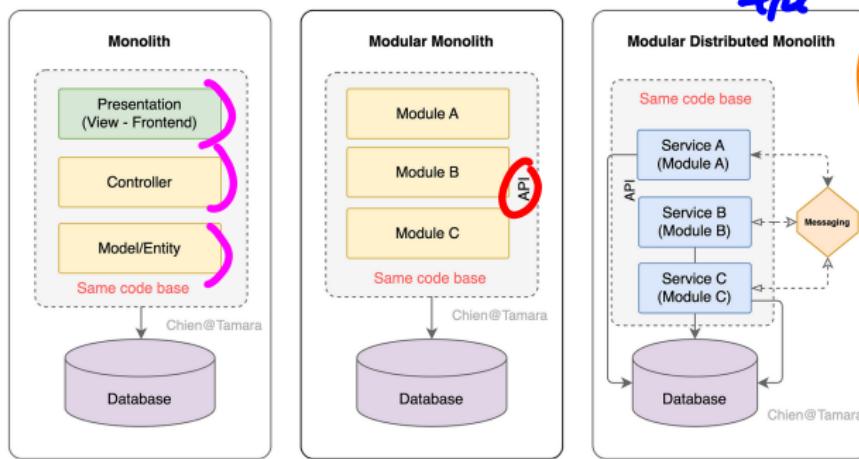
- There are several types of system architectures that are *used* in systems development.
- They include monolithic, client-server, peer-to-peer, and distributed architectures.
- Each type of architecture has its own advantages and disadvantages that *depend* on the specific requirements of the system.



Monolithic System Architecture

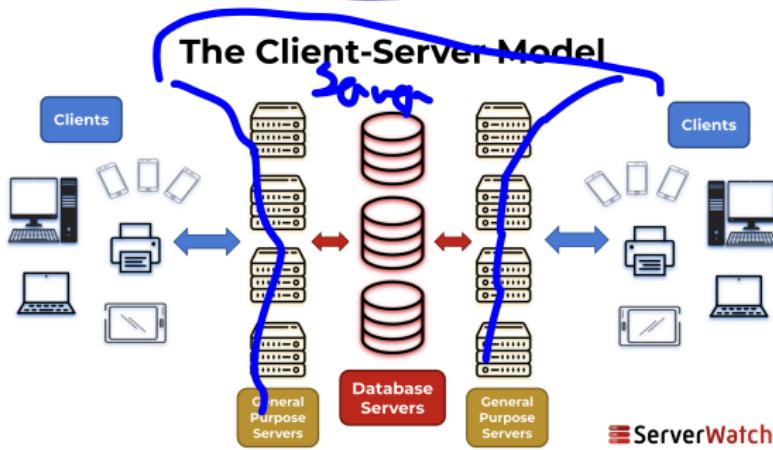
- A **monolithic system architecture** is a single-tier architecture that consists of a single unit that performs all the functions of the system.
- It is simple, easy to develop, and maintain, but it is not scalable and flexible. It is typically used for small systems that do not require high performance or reliability.

Java
sr C
.:
gui
data

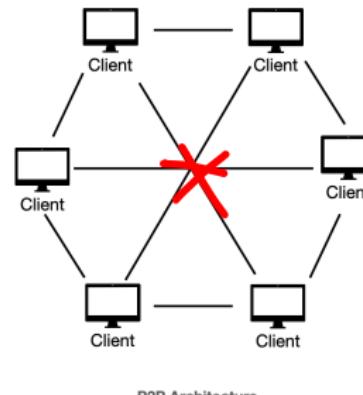
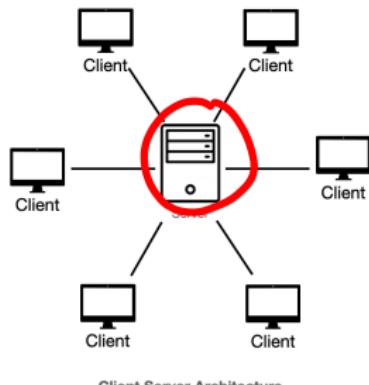
Client-Server System Architecture

- A **client-server system architecture** is a **two-tier architecture** that **consists** of a **client** and a **server** that **communicate** with each other over a **network**.
- It is **scalable**, **flexible**, and **efficient**, but it is **complex** and **difficult** to **develop** and **maintain**. It is used for **medium to large systems** that require **high performance** and **reliability**.



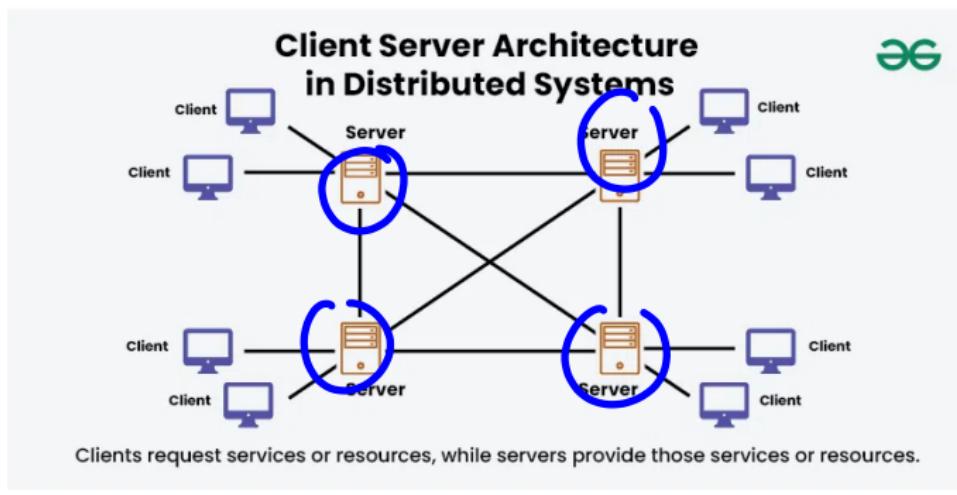
Peer-to-Peer System Architecture

- A **peer-to-peer system architecture** is a **two-tier architecture** that consists of a network of **peers** that communicate with each other directly.
- It is **scalable**, **flexible**, and **efficient**, but it is **complex** and **difficult to develop** and **maintain**. It is used for **medium to large systems** that require **high performance** and **reliability**.



Distributed System Architecture

- A **distributed system architecture** is a **multi-tier architecture** that consists of a **network of nodes** that communicate with each other over a **network**.
- It is **scalable**, **flexible**, and **efficient**, but it is **complex** and difficult to **develop and maintain**. It is used for **large systems** that require **high performance** and **reliability**.



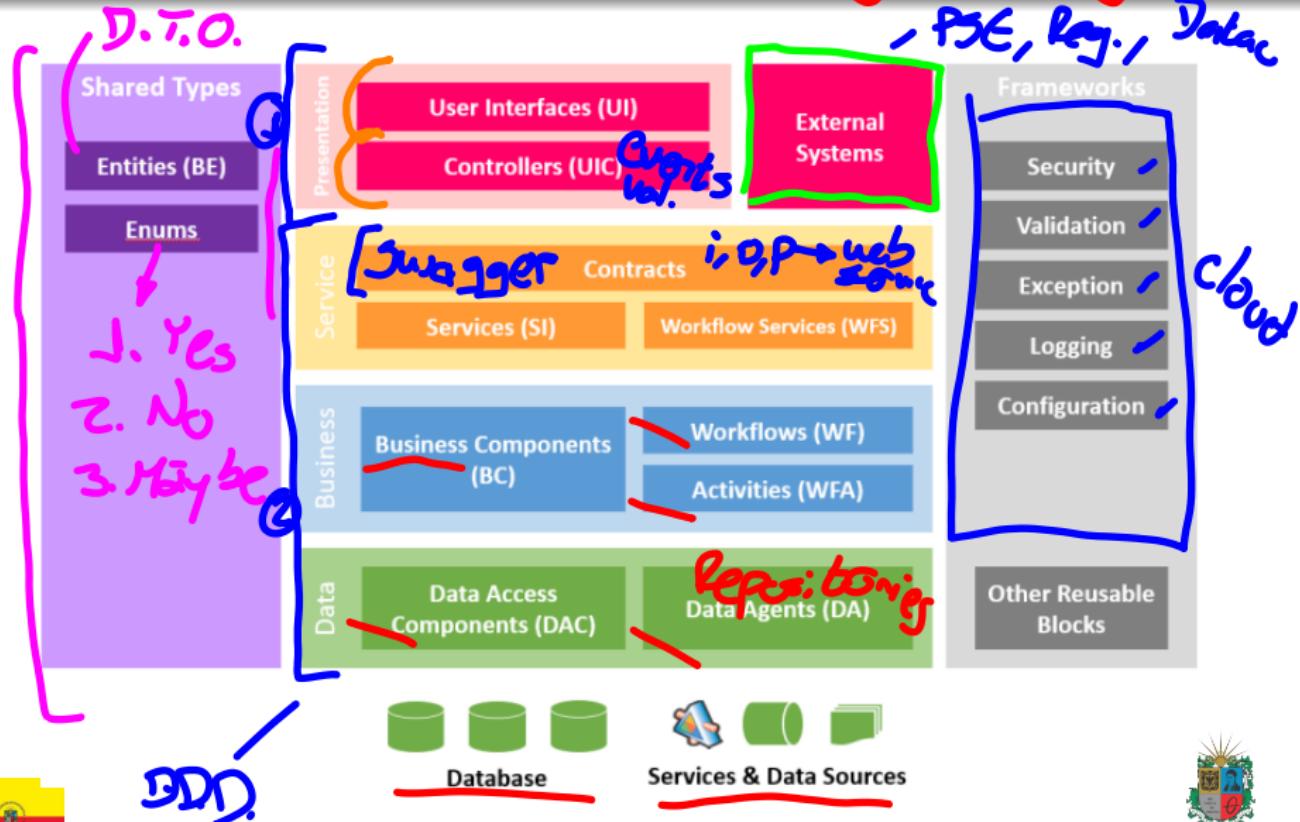
Software Architectures

- A **software architecture** is a high-level structure of a software system that defines its components, interactions, and relationships.
- A **software architecture** is the blueprint of a software system that guides its development and implementation.
- A **software architecture** is the foundation of a software system that ensures it meets the needs of its users.

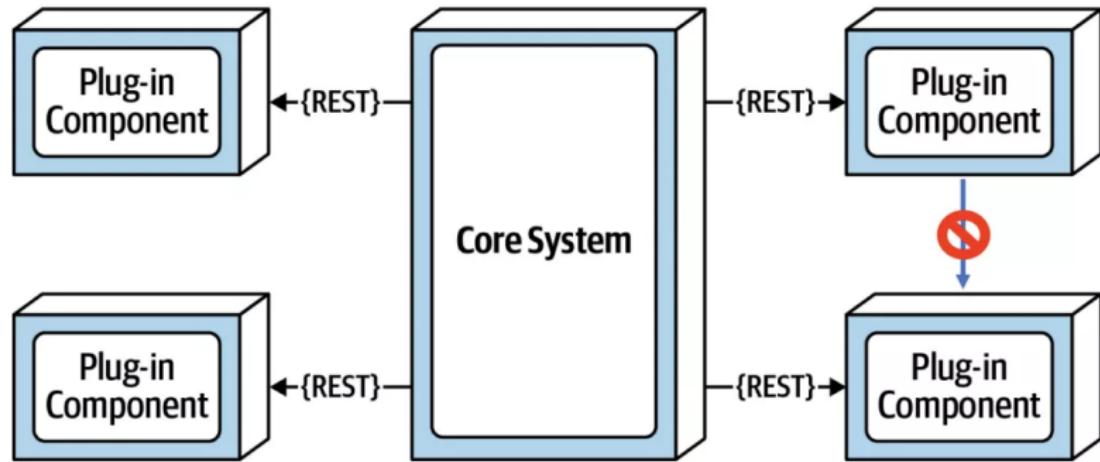
solve
of
problem



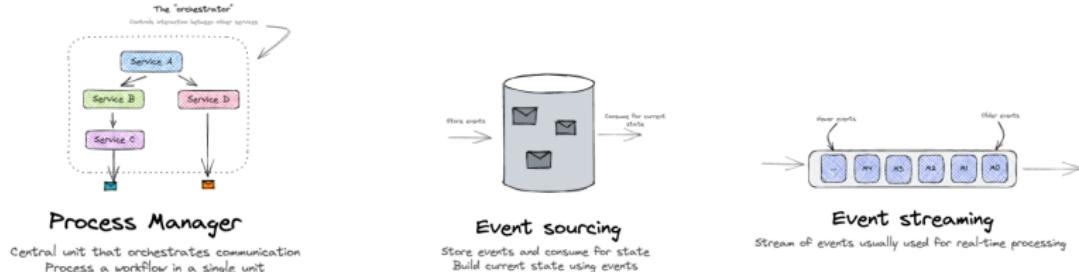
Layered Architecture Pattern → web ~ 2005



Microkernel Architecture Pattern



Event-based Architecture Pattern



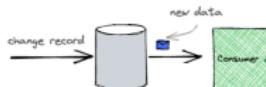
Inside event-driven architectures

What patterns will you come across when building event-driven architectures?



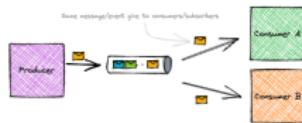
Point-to-point messaging (pull)

Messages put onto queue by sender
Receiver consumes messages from queue



Change data capture

Listening to events from changes in data
Consume events directly from DB

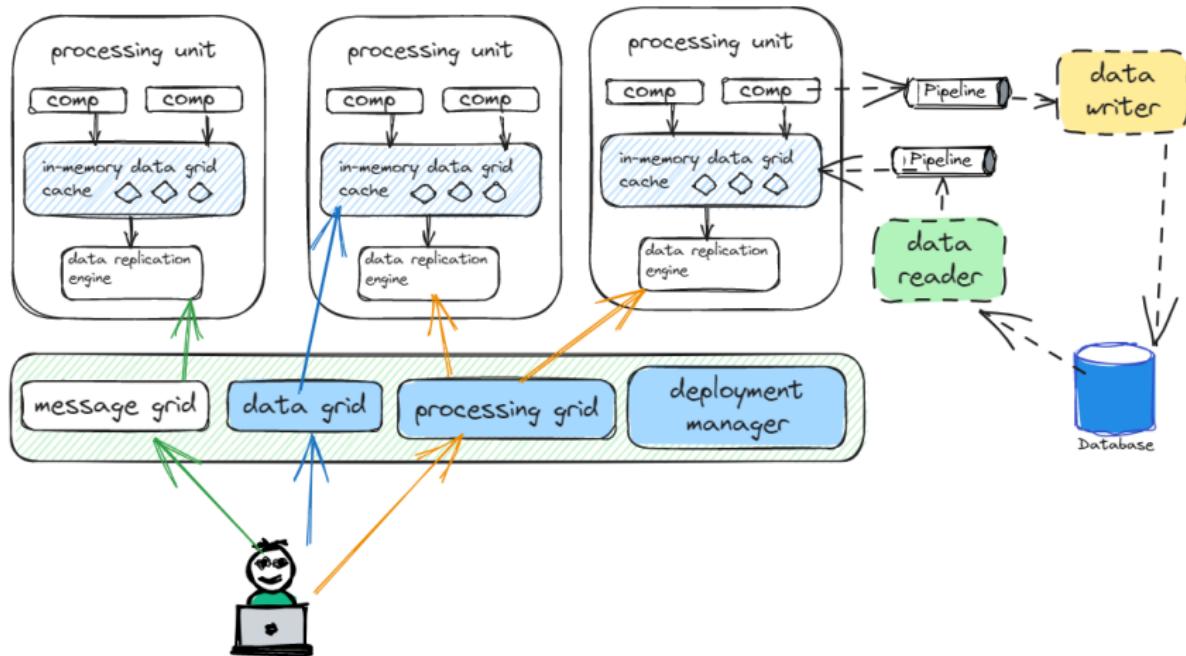


Pub/Sub (push)

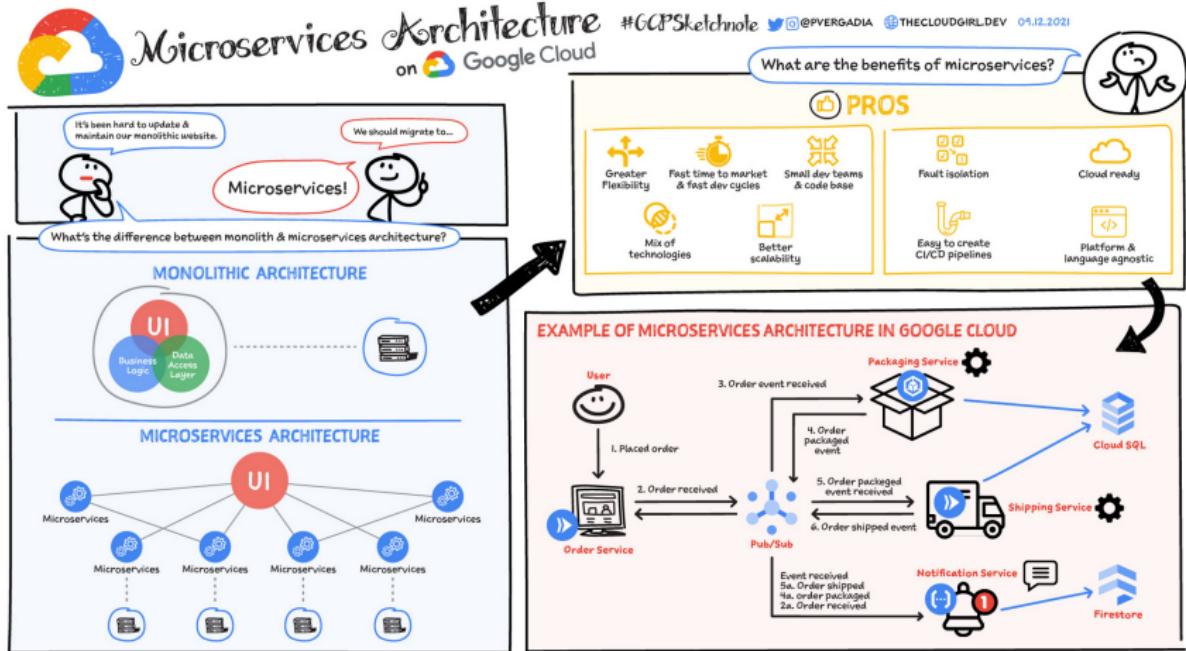
Publish messages/events to many subscribers
Each subscriber gets copy of event to process



Space-based Architecture Pattern

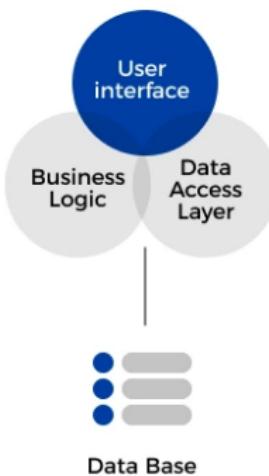


Microservices Architecture Pattern

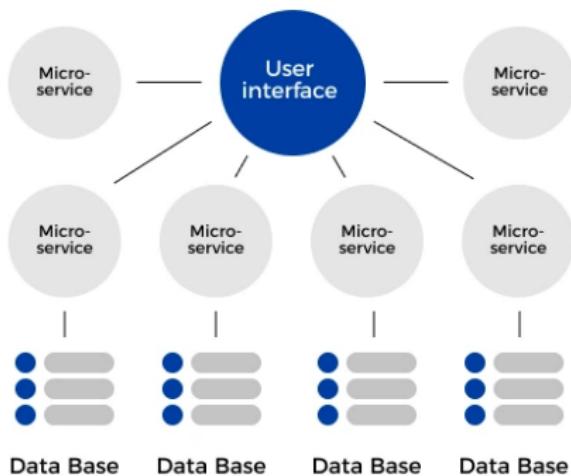


Monolithic vs. Microservices Architectures

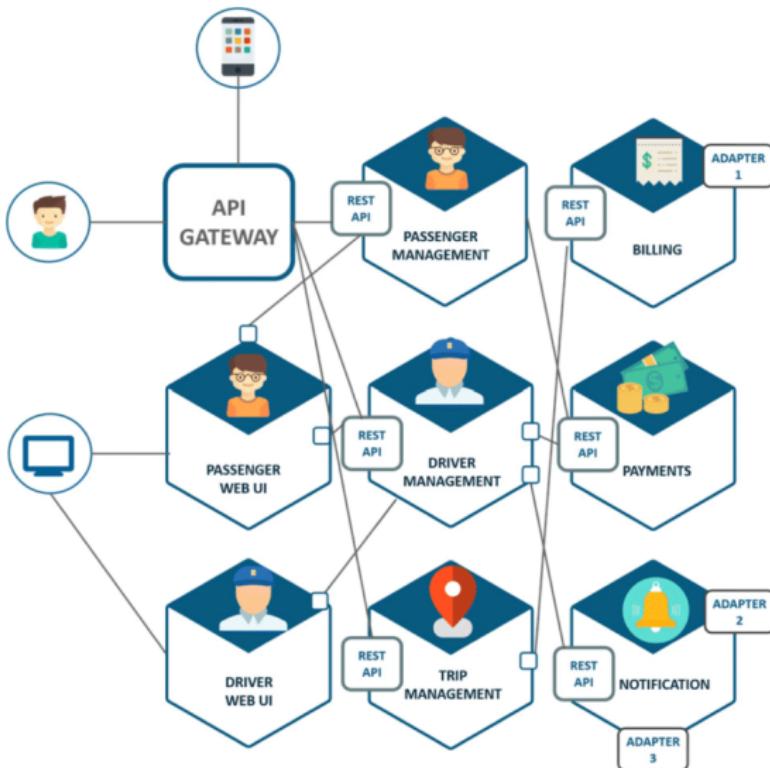
MONOLITHIC ARCHITECTURE



MICROSERVICE ARCHITECTURE



Case Study: Uber Microservices



Source: Kappagantula 2018



Outline

1 Systems Thinking

2 Reference Architectures



Thanks!

Questions?



Repo: www.github.com/EngAndres/ud-public/tree/main/courses/software_engineering_seminar

