Software Modeling Introduction

Software Modeling Foundations

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

Computer Engineer
Lecturer
Universidad Distrital Francisco José de Caldas

2024-III

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Basics of Software Development I

- The main idea is **solve real-world** problems based on **software solutions**. One of the main problems is the *complexity* of the systems, and to learn how to manage it.
- It is **not just to write code**, it is to have full software life cycle in mind, it means, to think in *design, tests, deploys, maintenance, a lot of tasks*.

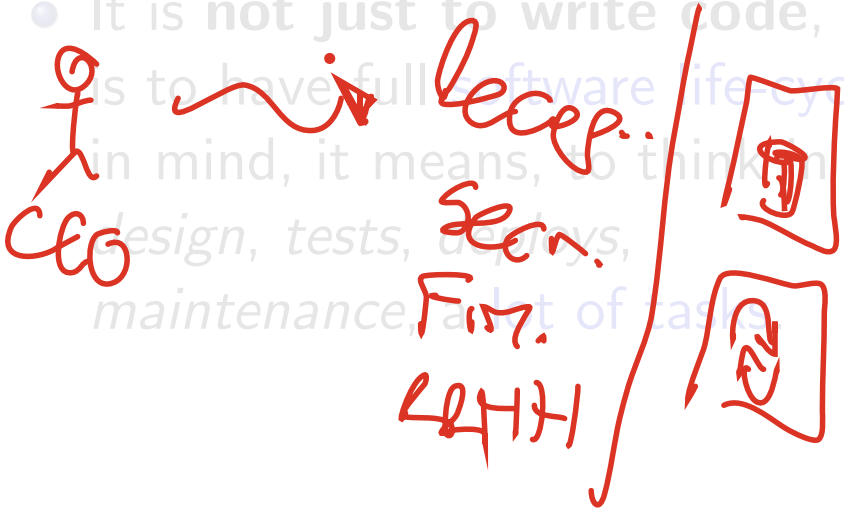


Figure: Prompt: Draw a python developer.

Basics of Software Development I

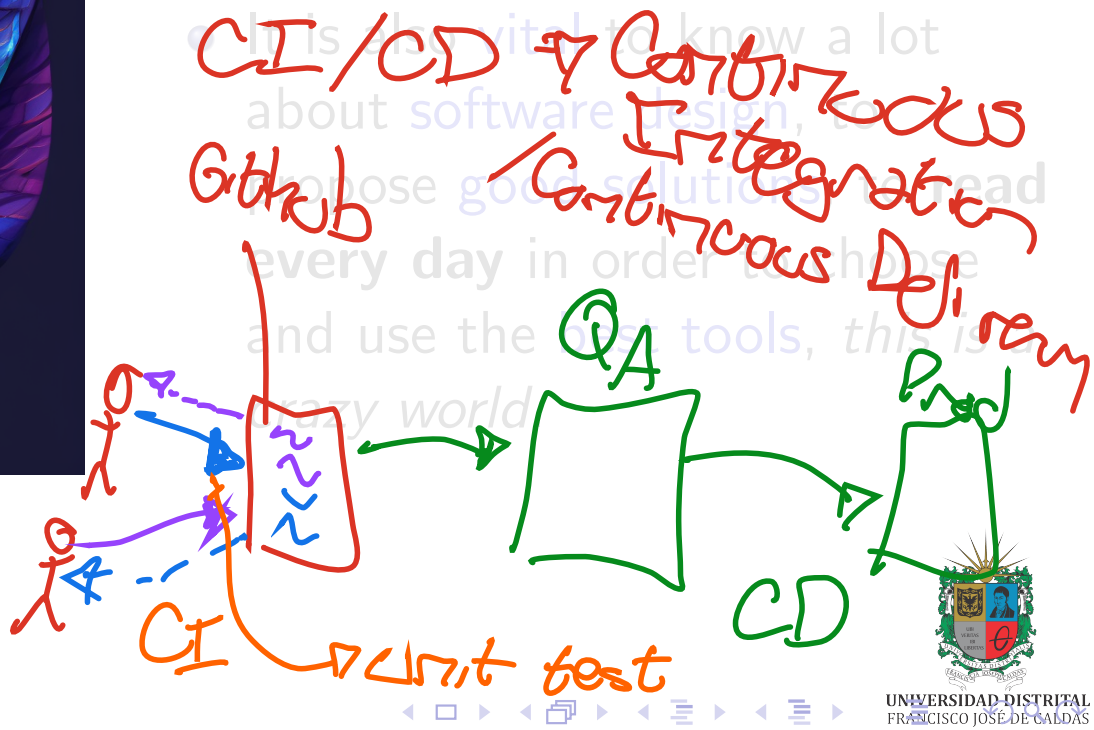
- The main idea is **solve real-world** problems based on **software solutions**. One of the main problems is the *complexity* of the systems, and to learn how to manage it.
- It is **not just to write code**, it is to have full software life-cycle in mind, it means, to think in *design*, *tests*, *deploys*, *maintenance*, a lot of tasks.

Figure: Prompt: Draw a python developer.

Basics of Software Development II

Figure: Prompt: Draw a python developer.

- However to **write code** is the most important task, and it is the main skill to have. You could write code to automate tests, deployments, integrations,. . . .

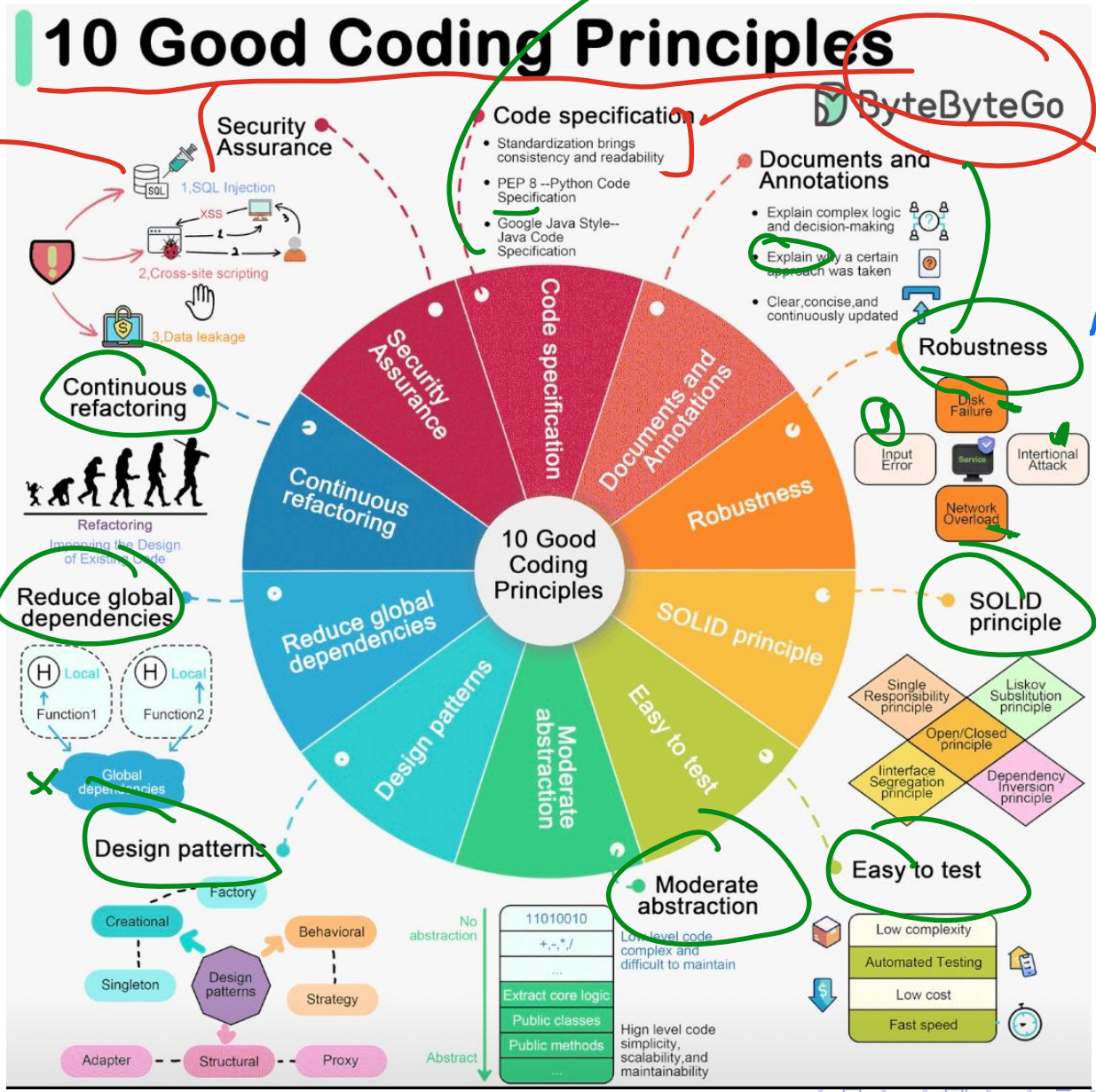


Basics of Software Development II

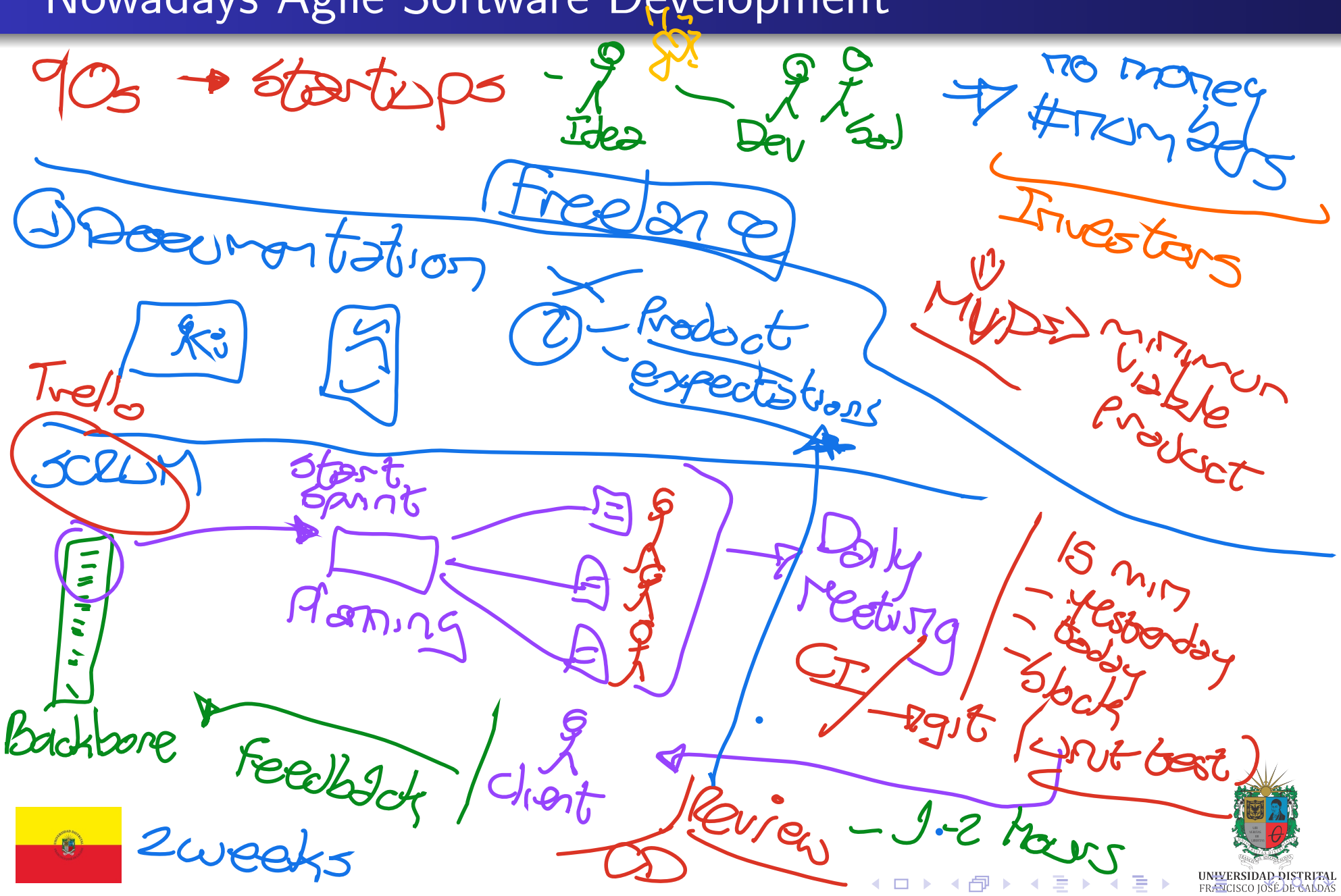
Figure: Prompt: Draw a python developer.

- However to **write code** is the most important task, and it is the main skill to have. You could write code to automate tests, deployments, integrations,. . . .
- It is also vital to know a lot about software design, to propose good solutions, to **read every day** in order to choose and use the best tools, *this is a crazy world*.

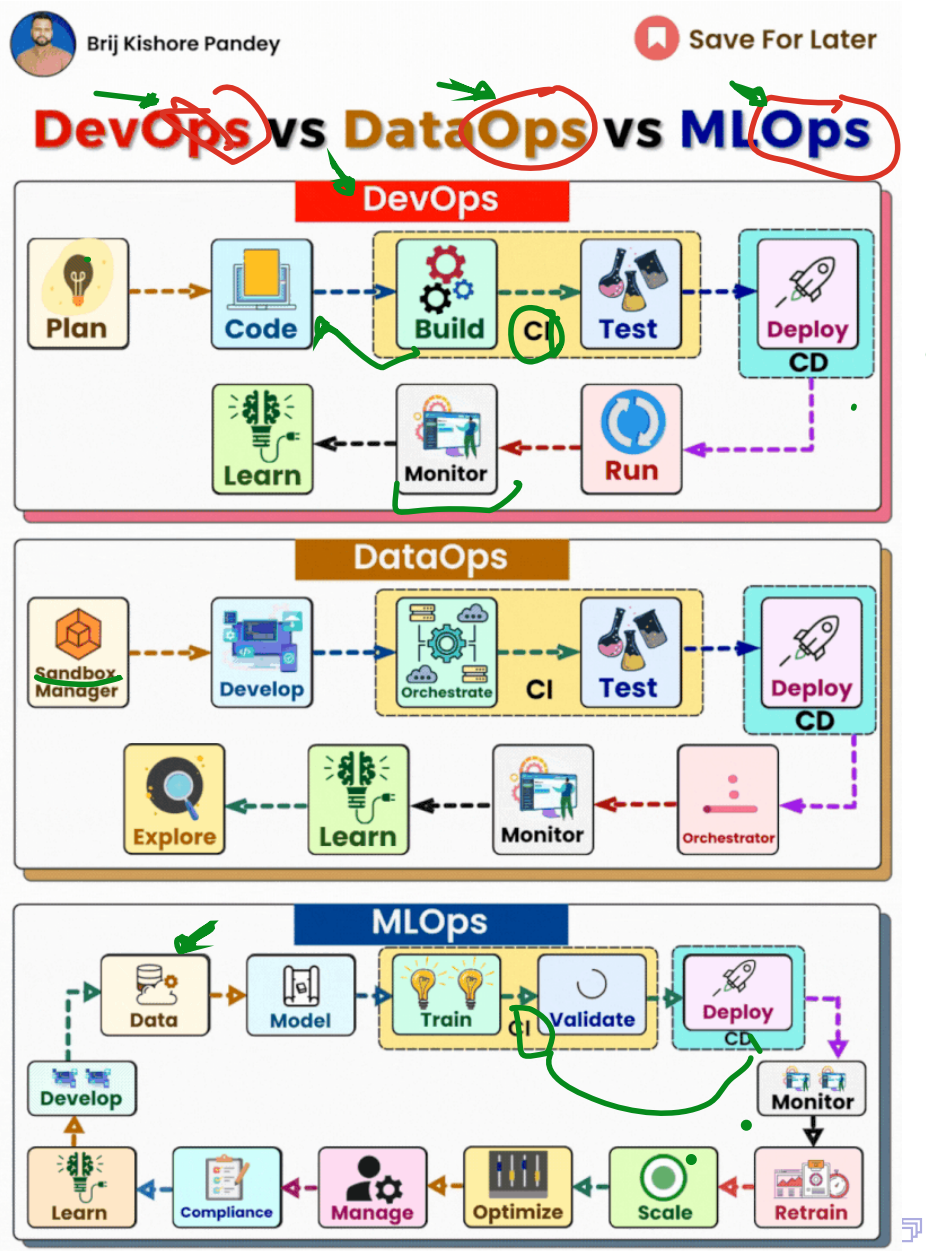
10 Good Coding Principles



Nowadays Agile Software Development



DataOps *Vs.* DevOps *Vs.* MLOps



Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Basics of Software Architecture I

- It is important to **develop** innovative and sophisticated **software** to provide a nice solution for **end users needs**.
- **Software architecture** brings innovation and good structure.
- The goal of **software architecture** is to minimize the human efforts required to build and maintain the required system.

Figure: Prompt: A python developer watching a building architecture draws.

Basics of Software Architecture I

- It is important to **develop** innovative and sophisticated **software** to provide a nice solution for **end users needs**.
- **Software architecture** brings innovation and robust structure.
- The goal of **software architecture** is to minimize the human efforts required to build and maintain the expected system.

Figure: Prompt: A python developer watching a building architecture draws.

Basics of Software Architecture I

- It is important to **develop** innovative and sophisticated **software** to provide a nice solution for **end users needs**.
- **Software architecture** brings innovation and robust structure.
- The goal of **software architecture** is to minimize the human efforts required to build and maintain the expected system.

Figure: Prompt: A python developer watching a building architecture draws.

Basics of Software Architecture II

Figure: Prompt: A python developer watching a building architecture draws.

- A **software architecture** is the skeleton for a complete software system. It leads the system to be **scalable**, **reliable**, and **maintainable**. Also it helps to take better technical decisions.
- There are some **software architecture styles**, each one with pros/cons, and specific use cases. However they try to provide a **reference solution** for a high-level structure of a software system.

80's 90's → 3 years

→ Components

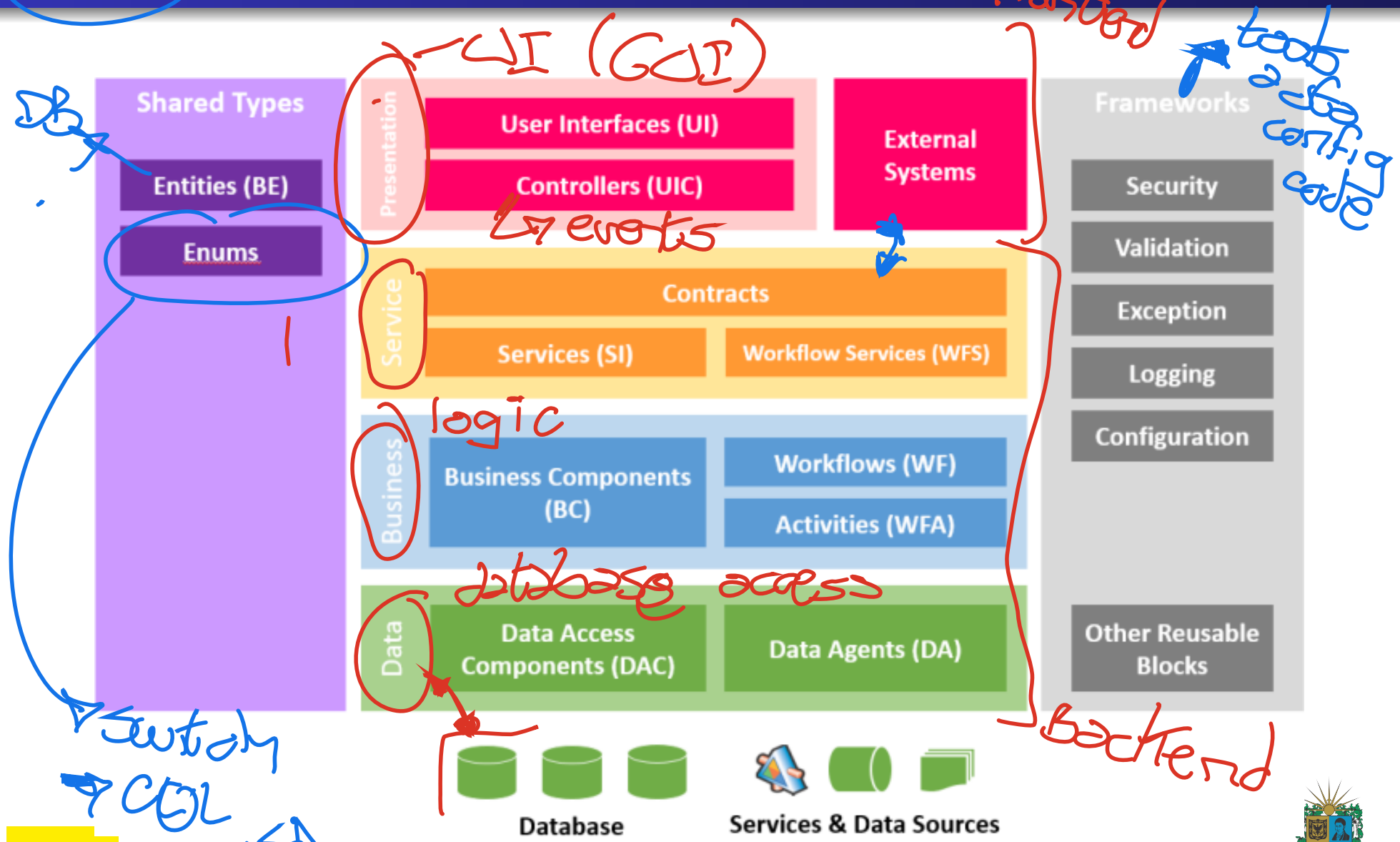
UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Basics of Software Architecture II

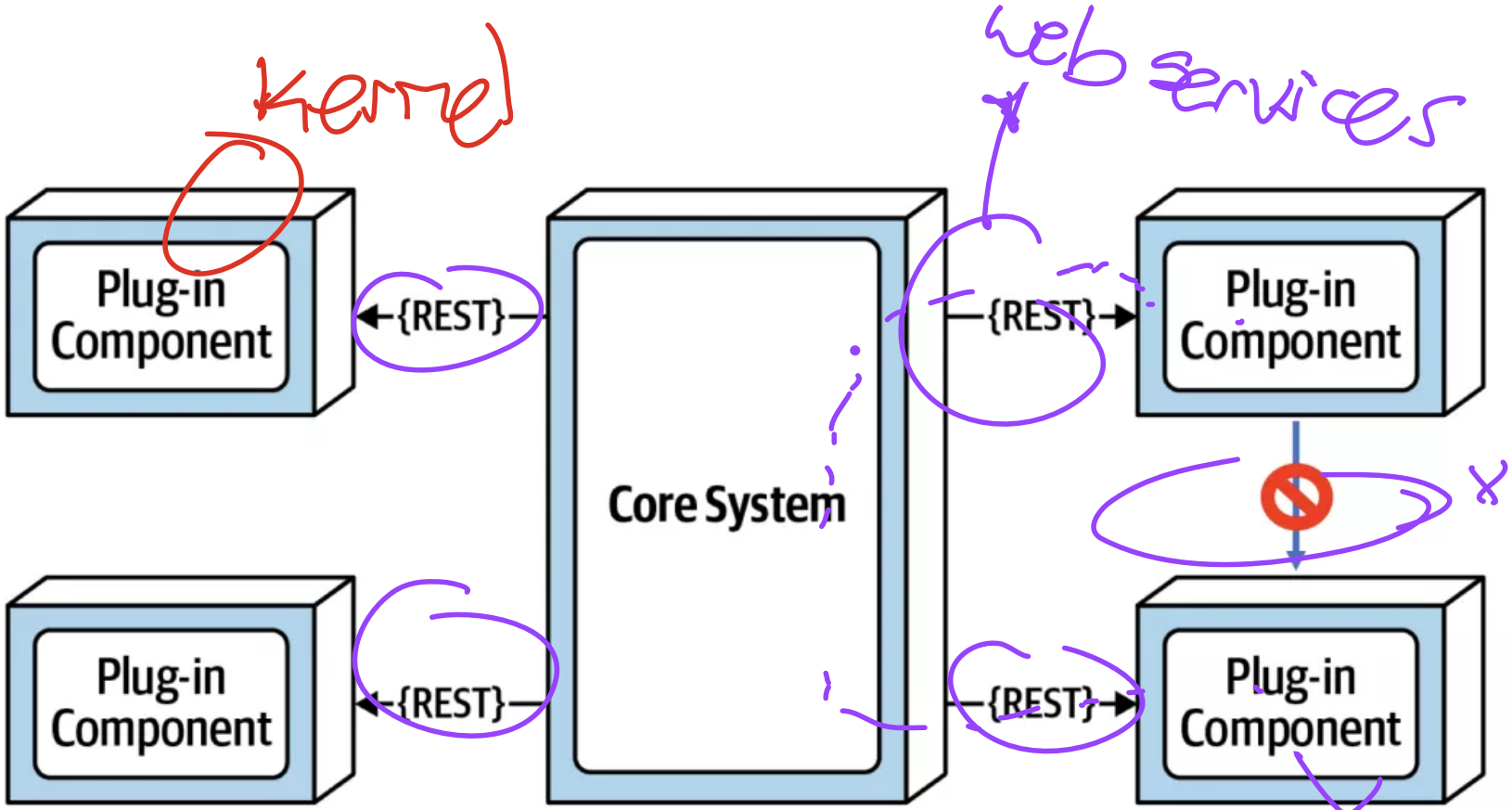
Figure: Prompt: A python developer watching a building architecture draws.

- A **software architecture** is the skeleton for a complete software system. It leads the system to be **scalable**, **reliable**, and **maintainable**. Also it helps to take better technical decisions.
- There are some **software architecture styles**, each one with pros/cons, and specific use cases. However, they try to provide a **reference solution** for a high-level structure of a software system.

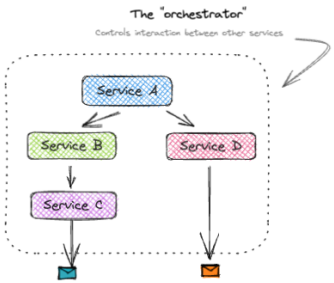
Layered Architecture Pattern



Microkernel Architecture Pattern

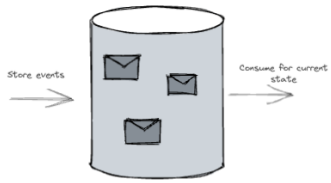


Event-based Architecture Pattern



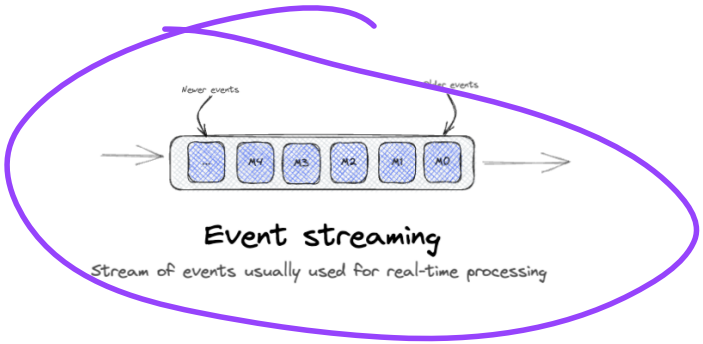
Process Manager

Central unit that orchestrates communication
Process a workflow in a single unit



Event sourcing

Store events and consume for state
Build current state using events



Event streaming

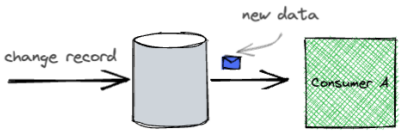
Stream of events usually used for real-time processing

Inside event-driven architectures

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

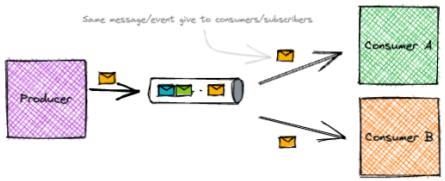
Point-to-point messaging (pull)

Messages put onto queue by sender
Receiver consumes messages from queue



Change data capture

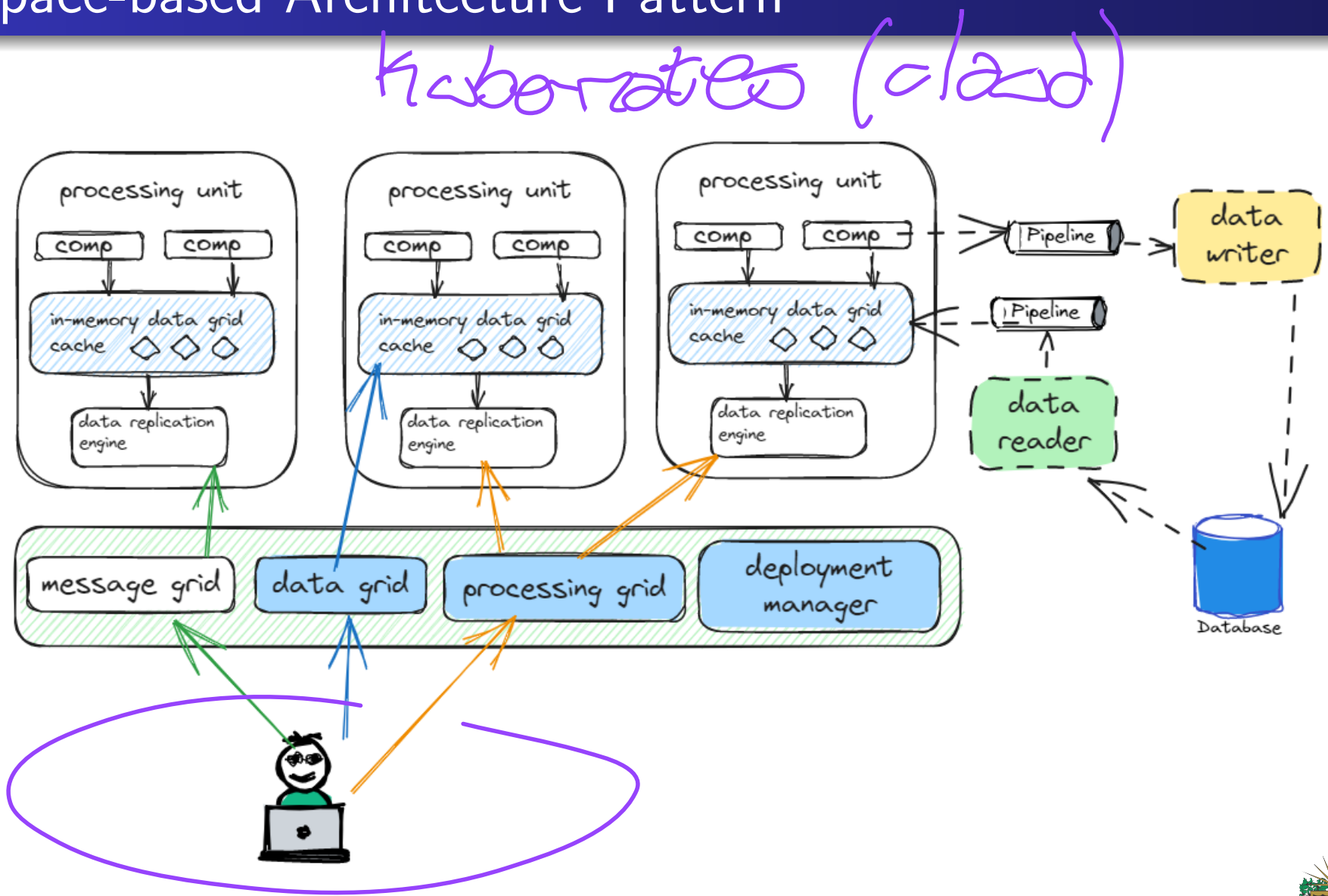
Listening to events from changes in data
Consume events directly from DB



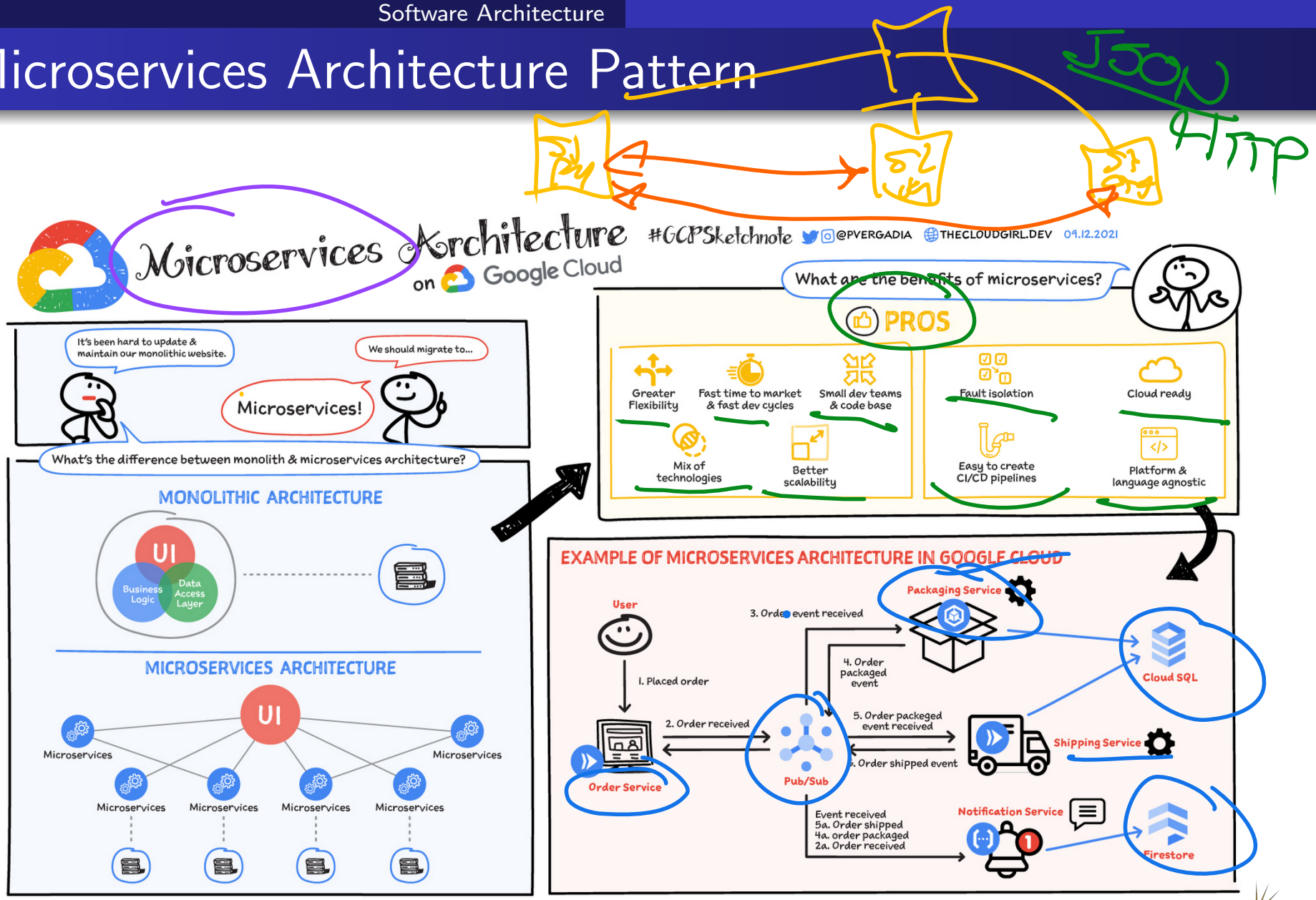
Pub/Sub (push)

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

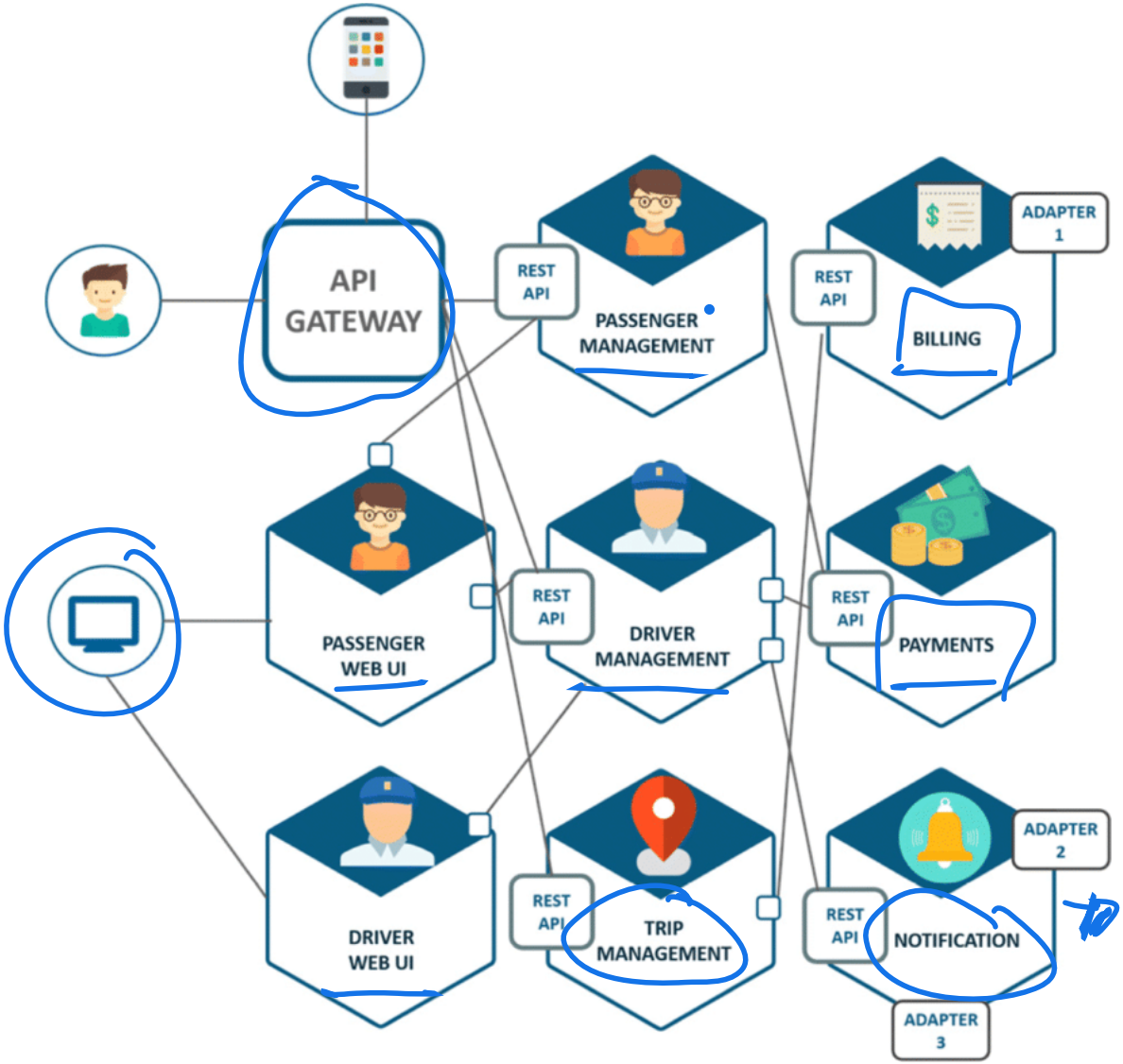
Space-based Architecture Pattern



Microservices Architecture Pattern



Case of Study: Microservices of Uber



Source: Kappagantula 2018

Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Basics of Object-Oriented Design I

- **Object-oriented** has become one of the most traditional and popular paradigms in software development.
- It is based on the concept of **objects**, which can contain data, in the form of fields (often known as *attributes* or *properties*), and code, in the form of procedures (often known as *methods*).

Figure: Prompt: Make an image of different real-world objects with binary inside each one.

Basics of Object-Oriented Design I

- **Object-oriented** has become one of the most traditional and popular paradigms in software development.
- It is based on the concept of **objects**, which can contain data, in the form of fields (often known as *attributes* or *properties*), and code, in the form of procedures (often known as *methods*).

Figure: Prompt: Make an image of different real-world objects with binary inside each one.

Basics of Object-Oriented Design II

Figure: Prompt: Make an image of different real-world objects with binary inside each one.

- The idea is to design a **system modularly**, and to make it easier to maintain, and to understand. Also the idea is emphasize the **reuse of code**.
- The main principles of **OOD** are:
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

Basics of Object-Oriented Design II

Figure: Prompt: Make an image of different real-world objects with binary inside each one.

- The idea is to design a **system modularly**, and to make it easier to maintain, and to understand. Also the idea is emphasize the **reuse of code**.
- The main principles of **OOD** are:
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Encapsulation in OOD

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

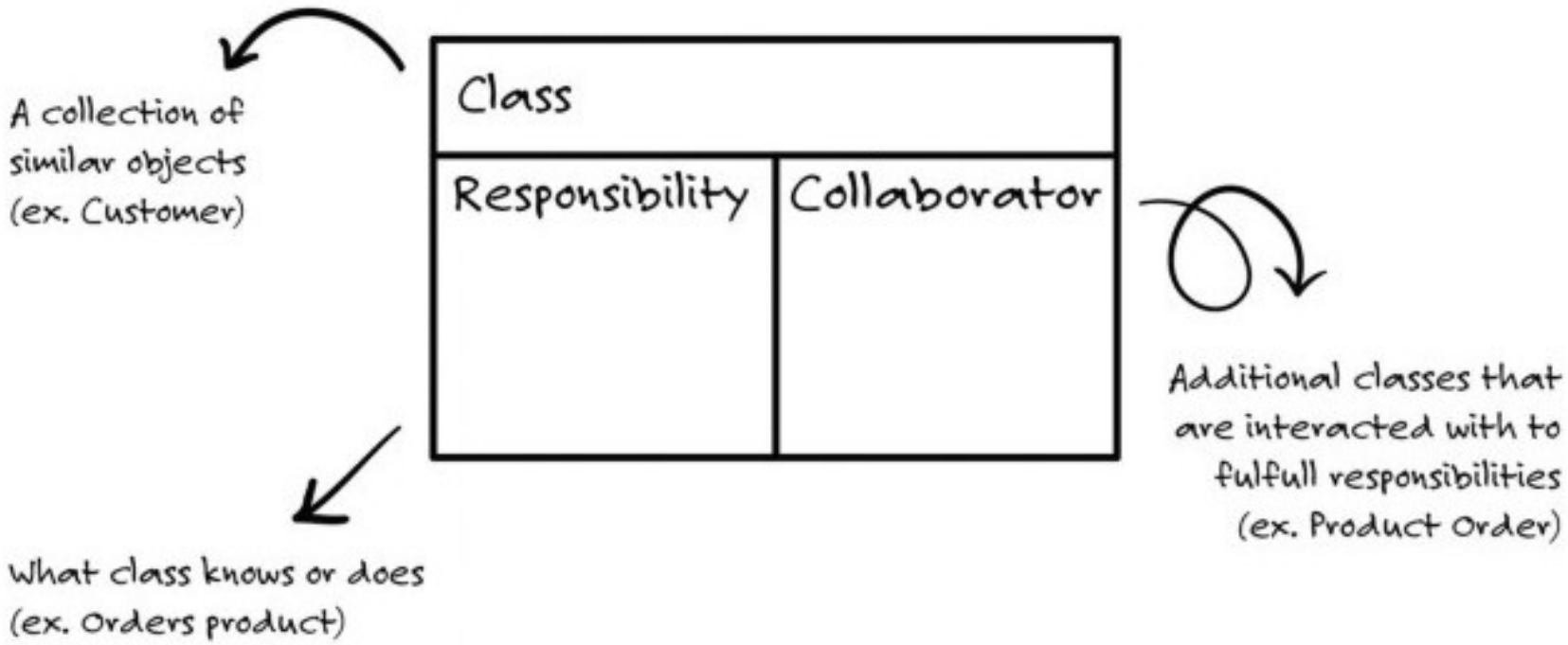
Abstraction in OOD

Inheritance in OOD

Polymorphism in OOD

Class-Responsability-Collaboration Cards (CRC)

The **CRC cards** are a brainstorming tool used in the design of object-oriented software.



Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Basics of Domain-Driven Design I

- **DDD** is focusing on the **core domain** and domain logic, it is a way of thinking aimed at accelerating software projects that have to deal with complicated domains.
- The essential terms of **DDD** are *context*, *model*, *ubiquitous language*, *bounded context*, and *business logic in layers*.
- **DDD** is a set of principles and patterns that help to **design** a system ensuring alignment with the real-world business needs.

Figure: Prompt: Draw a soccer coach teaching robots soccer players.

Basics of Domain-Driven Design I

- **DDD** is focusing on the **core domain** and domain logic, it is a way of thinking aimed at accelerating software projects that have to deal with complicated domains.
- The essential terms of **DDD** are *context*, *model*, *ubiquitous language*, *bounded context*, and *business logic in layers*.
- **DDD** is a set of principles and patterns that help to **design** a system ensuring alignment with the real-world business needs.

Figure: Prompt: Draw a soccer coach teaching robots soccer players.

Basics of Domain-Driven Design I

- **DDD** is focusing on the **core domain** and domain logic, it is a way of thinking aimed at accelerating software projects that have to deal with complicated domains.
- The essential terms of **DDD** are *context*, *model*, *ubiquitous language*, *bounded context*, and *business logic in layers*.
- **DDD** is a set of principles and patterns that help to **design** a system ensuring alignment with the real-world business needs.

Figure: Prompt: Draw a soccer coach teaching robots soccer players.

Basics of Domain-Driven Design II

Figure: Prompt: Draw a soccer coach teaching robots soccer players.

- The main principles of **DDD** are:
  - Focus on the core domain.
  - Base complex designs on models of the domain.
  - Constantly collaborate with domain experts.
  - Develop a knowledge-rich model.
- The **business logic** in layers is showed as follows:
  - Domain layer
  - Application layer
  - Presentation layer
  - Infrastructure layer

Basics of Domain-Driven Design II

Figure: Prompt: Draw a soccer coach teaching robots soccer players.

- The main principles of **DDD** are:
  - Focus on the core domain.
  - Base complex designs on models of the domain.
  - Constantly collaborate with domain experts.
  - Develop a knowledge-rich model.
- The **business logic** in layers is showed as follows:
  - Domain layer
  - Application layer
  - Presentation layer
  - Infrastructure layer

Basics of Domain-Driven Design II

Figure: Prompt: Draw a soccer coach teaching robots soccer players.

- The main principles of **DDD** are:
  - Focus on the core domain.
  - Base complex designs on models of the domain.
  - Constantly collaborate with domain experts.
  - Develop a knowledge-rich model.
- The **business logic** in layers is showed as follows:
  - Domain Layer.
  - Application Layer.
  - Presentation Layer.
  - Infrastructure Layer.

Basics of Domain-Driven Design II

Figure: Prompt: Draw a soccer coach teaching robots soccer players.

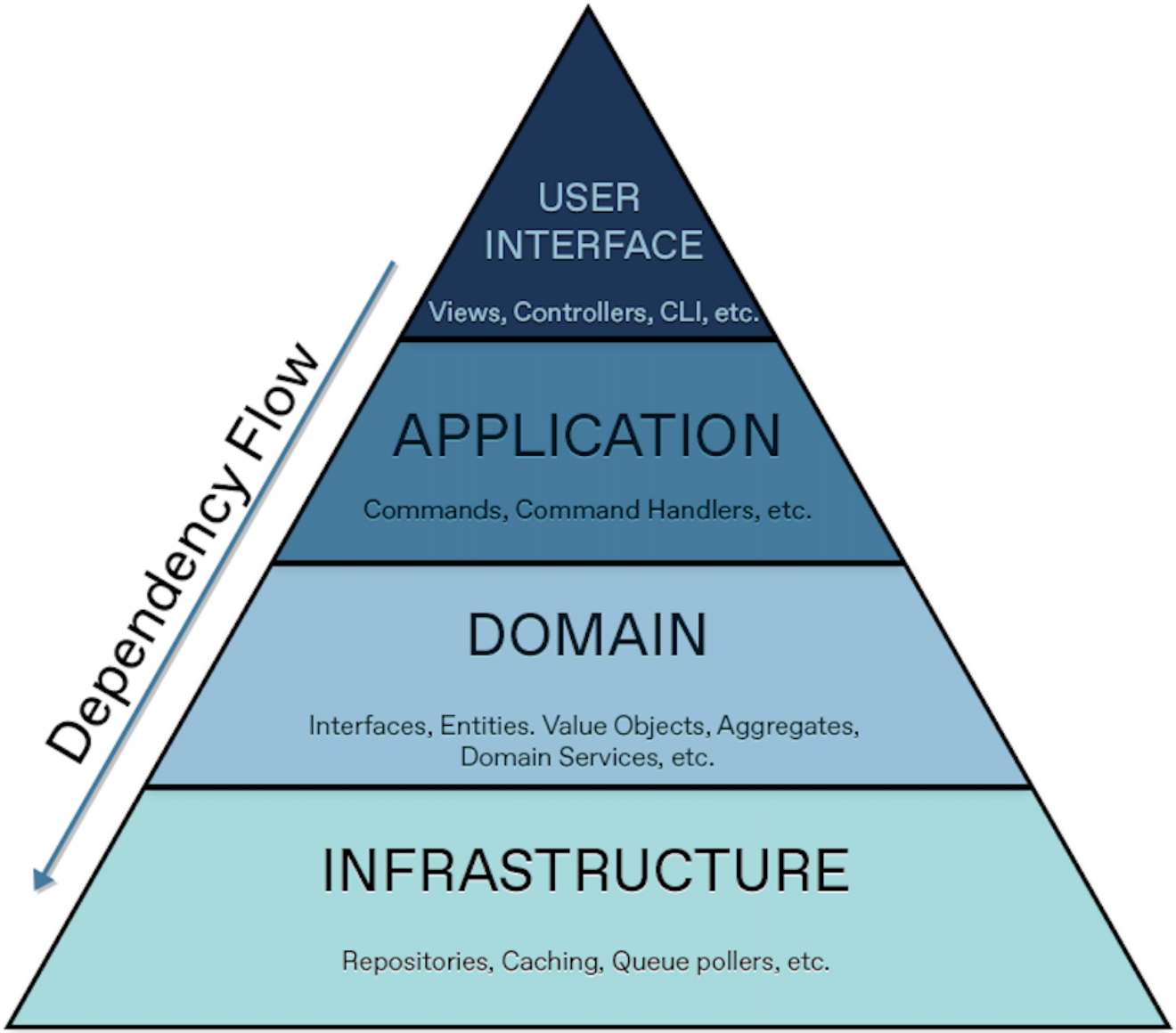
- The main principles of **DDD** are:
  - Focus on the core domain.
  - Base complex designs on models of the domain.
  - Constantly collaborate with domain experts.
  - Develop a knowledge-rich model.
- The **business logic** in layers is showed as follows:
  - Domain Layer.
  - Application Layer.
  - Presentation Layer.
  - Infrastructure Layer.

Basics of Domain-Driven Design II

Figure: Prompt: Draw a soccer coach teaching robots soccer players.

- The main principles of **DDD** are:
  - Focus on the core domain.
  - Base complex designs on models of the domain.
  - Constantly collaborate with domain experts.
  - Develop a knowledge-rich model.
- The **business logic** in layers is showed as follows:
  - Domain Layer.
  - Application Layer.
  - Presentation Layer.
  - Infrasctructure Layer.

Business Logic in Layers



Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Basics of Design Pattern I

- A **design pattern** is a practical proven solution to a recurring design problem in software engineering.
- A **design pattern** is not a finished design, just flexible or reusable parts of a complete solution.
- Created (or discovered) for **expert developers**, and had been matured by non-expert developers, they are like *food recipes*, or *play tik-tik-toe*.

Figure: Prompt: Draw a tik-tak-inning strategy.

Basics of Design Pattern I

- A **design pattern** is a practical proven solution to a recurring design problem in software engineering.
- A **design pattern** is not a finished design, just flexible or reusable parts of a complete solution.
- Created (or discovered) for **expert developers**, and had been matured by non-expert developers, they are like *food recipes*, or *play tik-tik-toe*.

Figure: Prompt: Draw a tik-tak-inning strategy.

Basics of Design Pattern I

- A **design pattern** is a practical proven solution to a recurring design problem in software engineering.
- A **design pattern** is not a finished design, just flexible or reusable parts of a complete solution.
- Created (or discovered) for **expert developers**, and had been matured by non-expert developers, they are like *food recipes*, or *play tik-tik-toe*.

Figure: Prompt: Draw a tik-tak-inning strategy.

Basics of Design Pattern II

Figure: Prompt: Draw a tik-tak-toe winning strategy.

- There are **23 design patterns**, classified in *three categories*: creational, structural, & behavioral patterns.
- Reported by the **Gang of Four** (**GoF**) in 1994, in the book Design Patterns: Elements of Reusable Object-Oriented Software.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Basics of Design Pattern II

Figure: Prompt: Draw a tik-tak-toe winning strategy.

- There are **23 design patterns**, classified in *three categories*: creational, structural, & behavioral patterns.
- Reported by the **Gang of Four** (**GoF**) in 1994, in the book Design Patterns: Elements of Reusable Object-Oriented Software.

Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Thanks!

Questions?

Repo: *https://github.com/EngAndres/ud-public/tree/main/courses/software-modeling*

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS