

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



Outline

- 1 Software Development
- 2 Software Architecture
- 3 Object-Oriented Design
- 4 Domain-Driven Design
- 5 Design Patterns



Outline

1 Software Development

2 Software Architecture

3 Object-Oriented Design

4 Domain-Driven Design

5 Design Patterns



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.

CEO
Sist. Inf.
Lecep.
Serc.
Fim.
LHHH



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**.

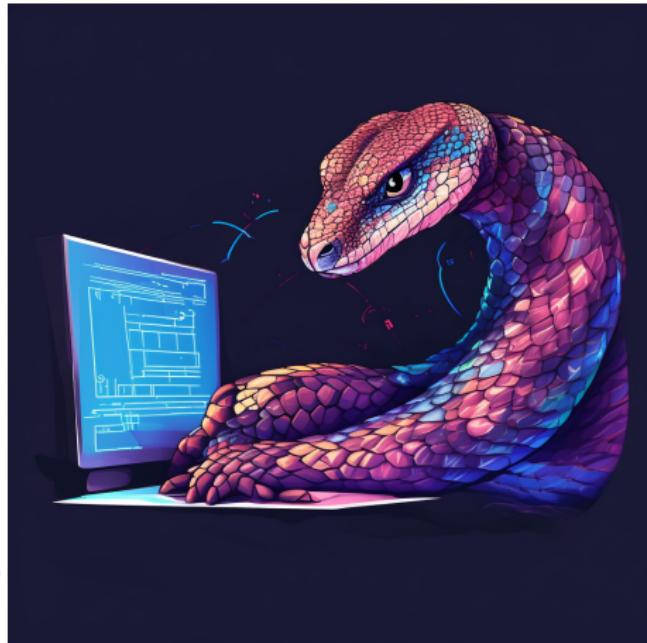
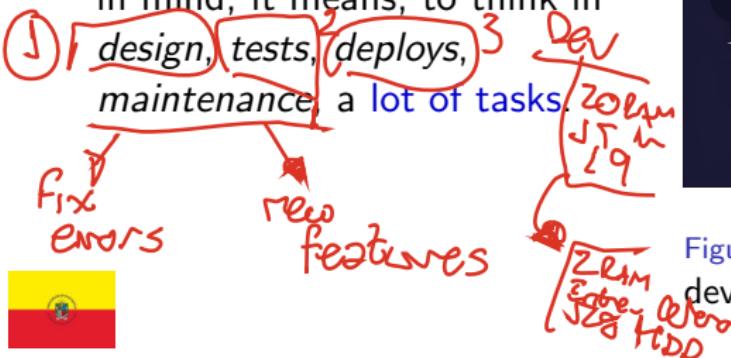


Figure: Prompt: Draw a python developer.



Basics of Software Development II

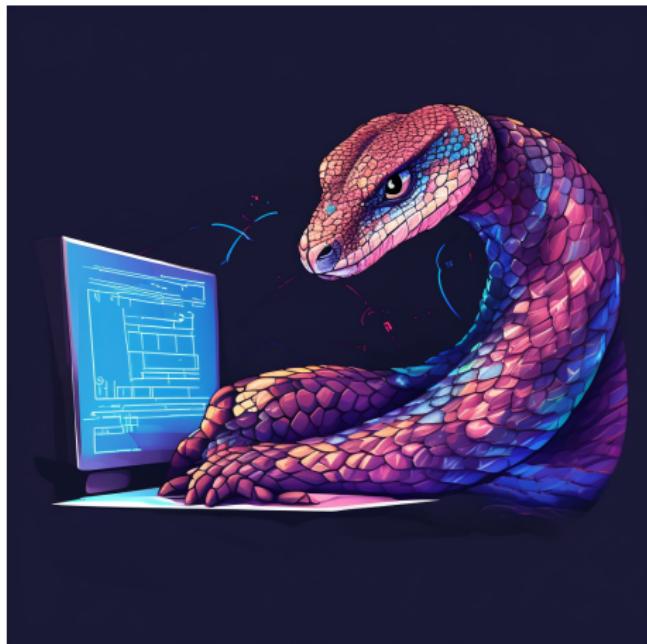
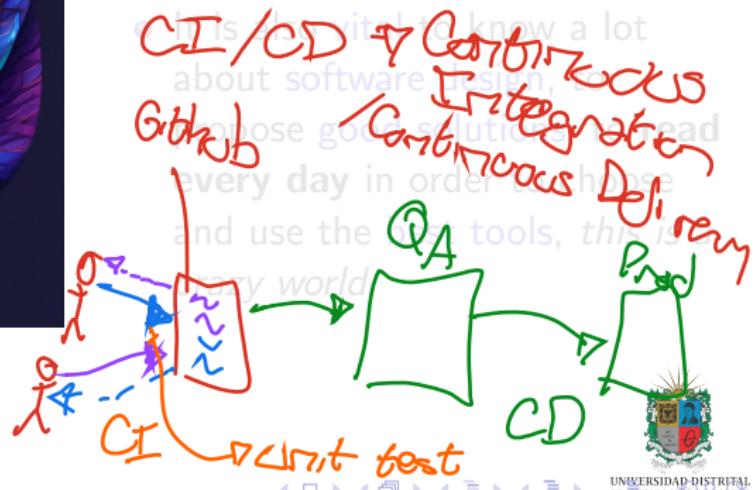


Figure: Prompt: Draw a python developer.



YAML

- 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

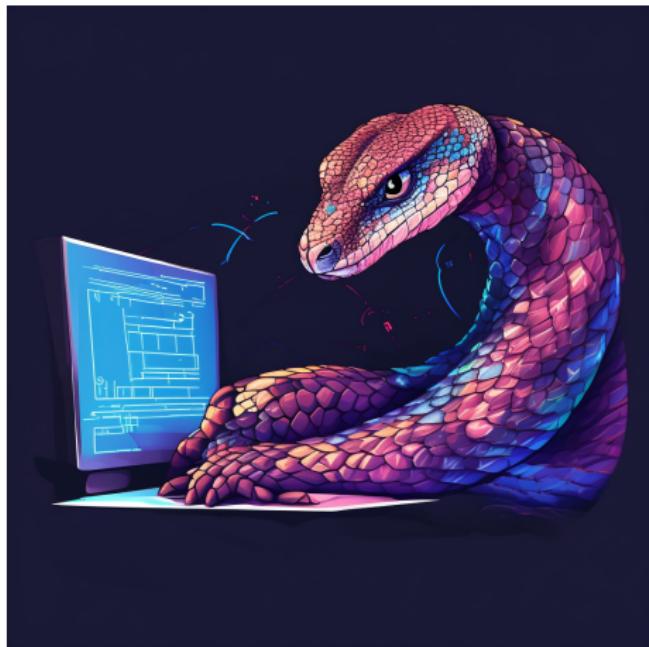


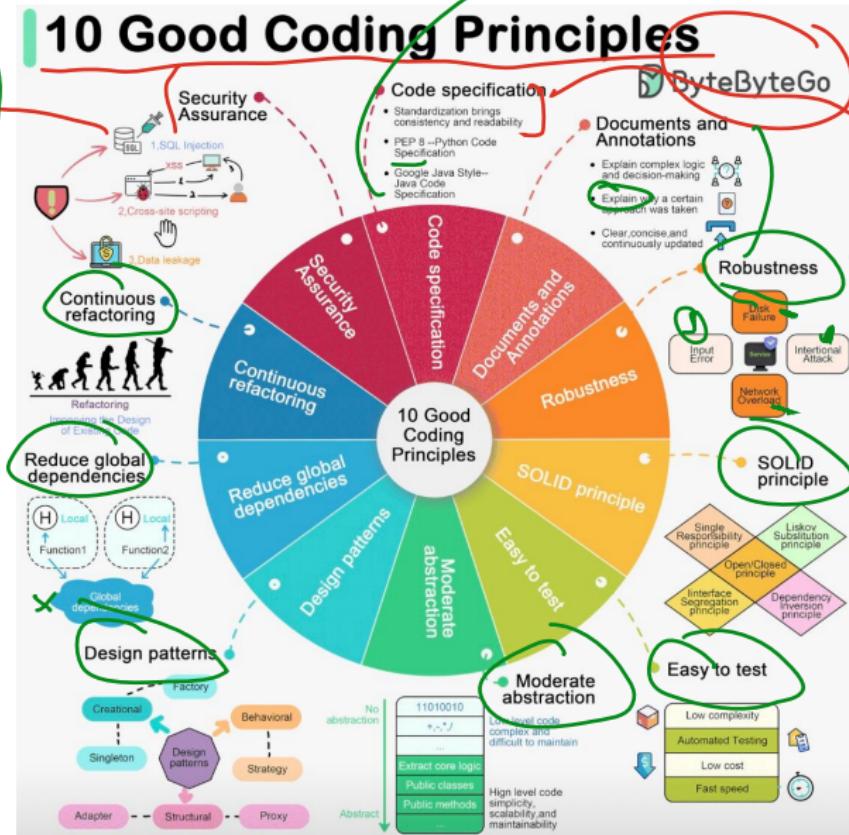
Figure: Prompt: Draw a python developer.



- 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,....** *medium*
- 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.*



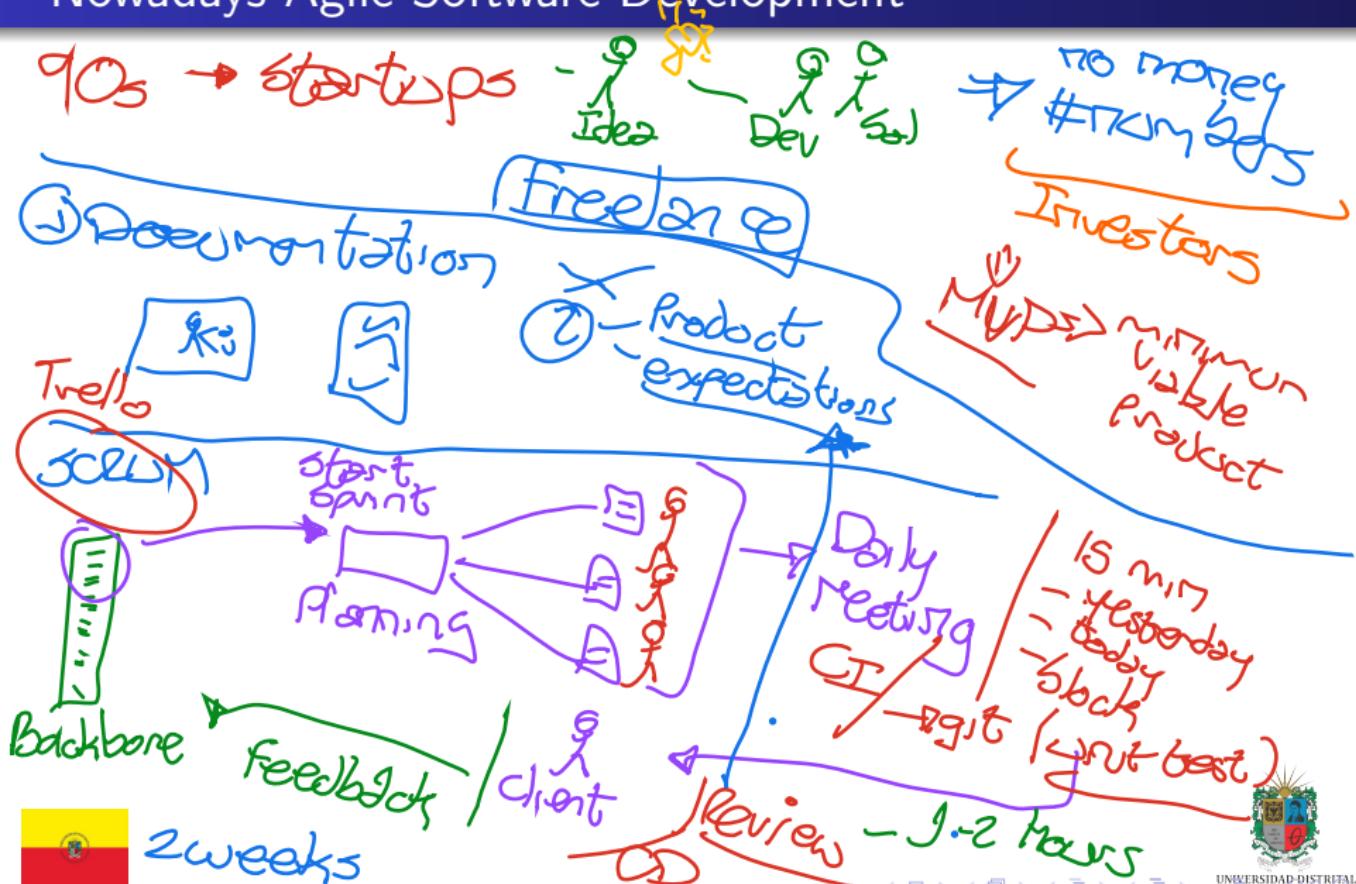
10 Good Coding Principles



Street
DB
ORM



Nowadays Agile Software Development



DataOps Vs. DevOps Vs. MLOps

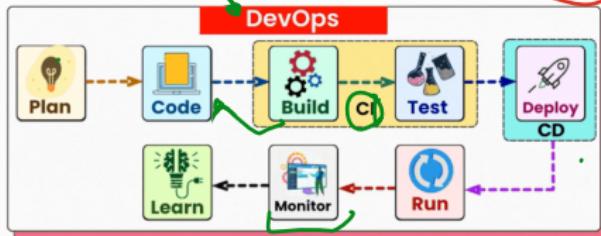


Brij Kishore Pandey

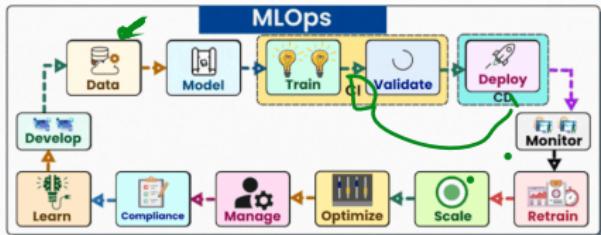
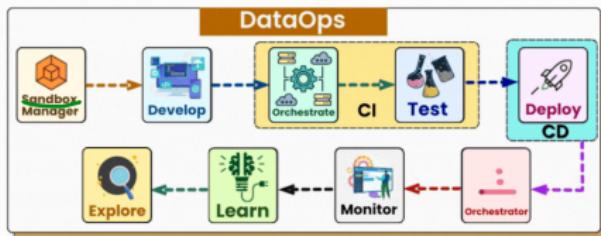


Save For Later

DevOps vs DataOps vs MLOps



Get
Feedback



Outline

- 1 Software Development
- 2 Software Architecture
- 3 Object-Oriented Design
- 4 Domain-Driven Design
- 5 Design Patterns



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 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.



Basics of Software Architecture II

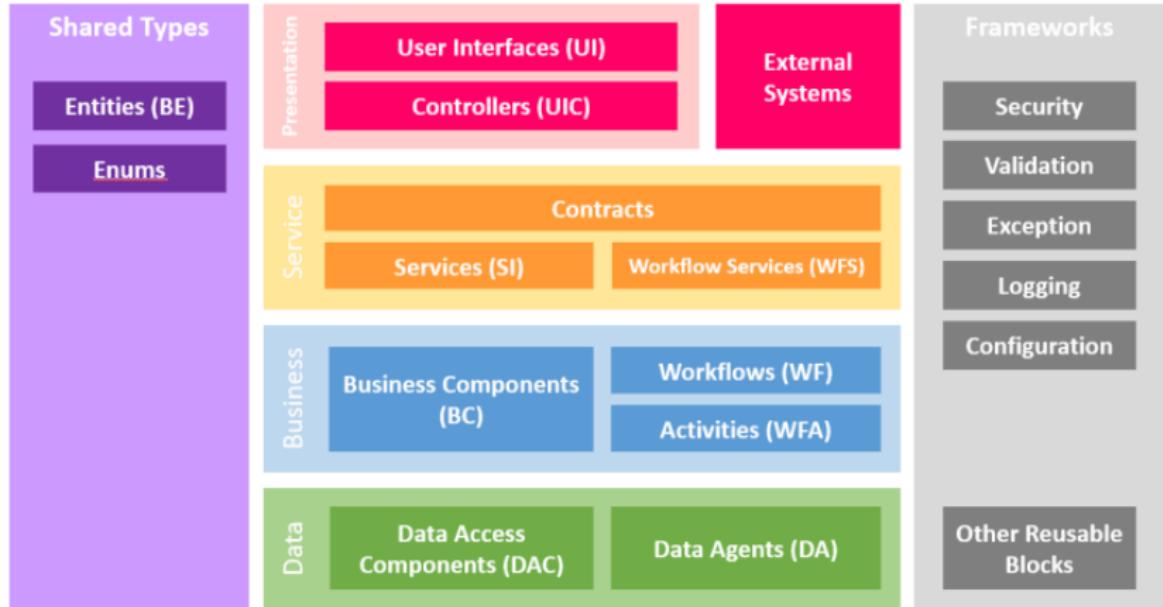


Figure: Prompt: A python developer watching a building architecture draws.

- 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



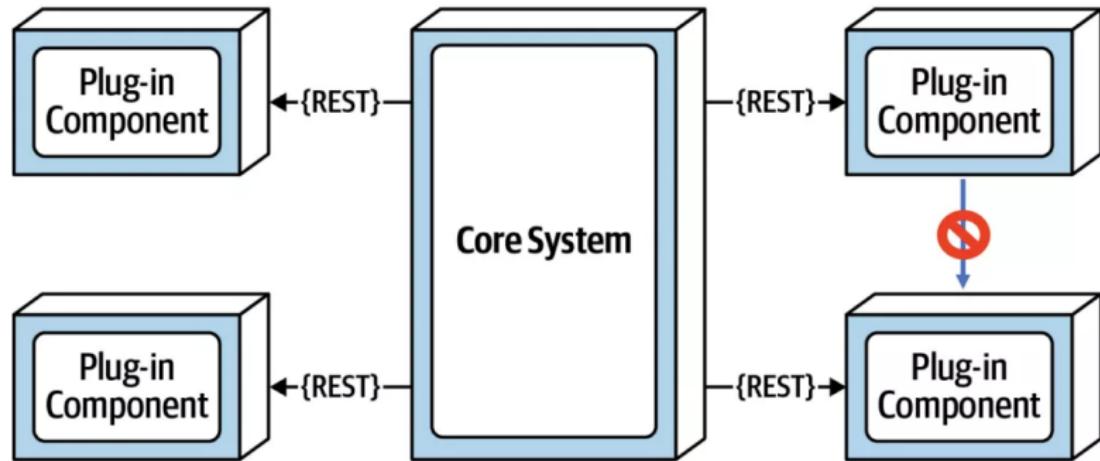
Database



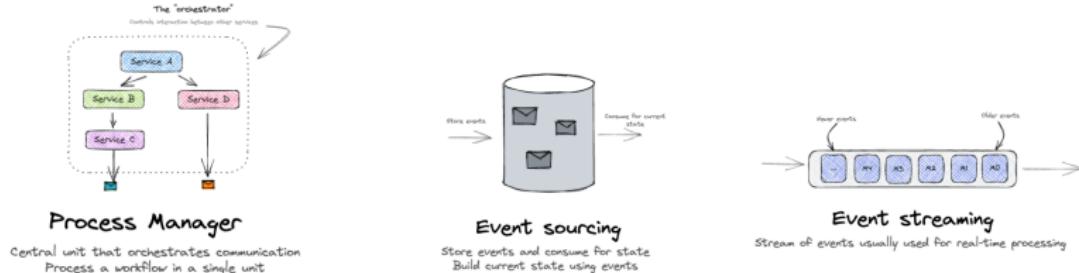
Services & Data Sources



Microkernel Architecture Pattern



Event-based Architecture Pattern



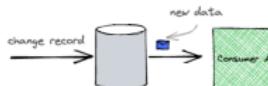
Inside event-driven architectures

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



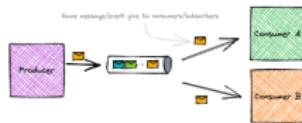
Point-to-point messaging (pull)

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

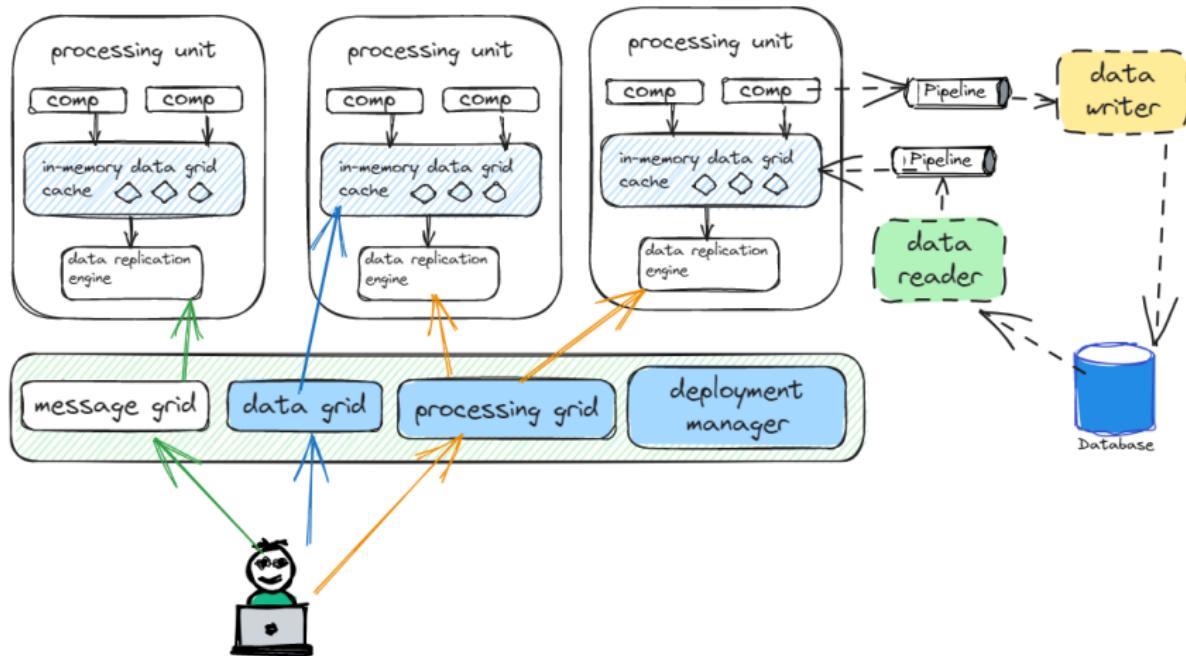


Pub/Sub (push)

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



Space-based Architecture Pattern



Microservices Architecture Pattern

Monolithic Architecture: A single large application where UI, Business Logic, and Data Access Layer are tightly coupled.

Microservices Architecture: A distributed system composed of small, independently deployable services. The UI is decoupled from the business logic and data access layer, which are further broken down into multiple microservices.

Benefits of Microservices (PROS):

- Greater Flexibility
- Fast time to market & fast dev cycles
- Small dev teams & code base
- Mix of technologies
- Better scalability
- Fault isolation
- Easy to create CI/CD pipelines
- Cloud ready
- Platform & language agnostic

EXAMPLE OF MICROSERVICES ARCHITECTURE IN GOOGLE CLOUD:

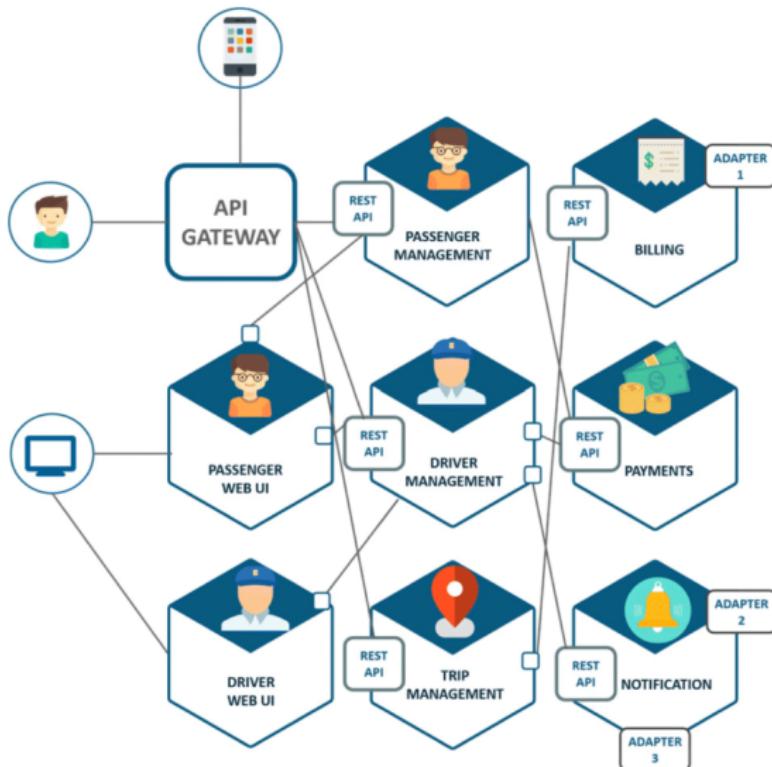
```

graph TD
    User((User)) -- "1. Placed order" --> OrderService[Order Service]
    OrderService -- "2. Order received" --> PubSub((Pub/Sub))
    PubSub -- "3. Order event received" --> PackagingService[Packaging Service]
    PackagingService -- "4. Order packaged event" --> CloudSQL[Cloud SQL]
    PackagingService -- "5. Order packaged event received" --> ShippingService[Shipping Service]
    ShippingService -- "6. Order shipped event" --> NotificationService[Notification Service]
    NotificationService -- "Event received: 5a. Order shipped, 5b. order packaged, 5c. Order received" --> Firestore[Firestore]
  
```

The diagram illustrates a workflow for placing an order. It starts with a User placing an order, which is received by an Order Service. The Order Service then sends a message to a Pub/Sub topic. From there, the message goes to a Packaging Service, which packages the order and sends it to Cloud SQL. The Packaging Service also receives a message back from the Order Service. Next, the Packaging Service sends a message to a Shipping Service, which then sends a message to a Notification Service. The Notification Service finally sends a message to a Firestore database.



Case of Study: Microservices of Uber



Source: Kappagantula 2018



Outline

- 1 Software Development
- 2 Software Architecture
- 3 Object-Oriented Design
- 4 Domain-Driven Design
- 5 Design Patterns



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

Encapsulation in OOD



Abstraction in OOD



Inheritance in OOD

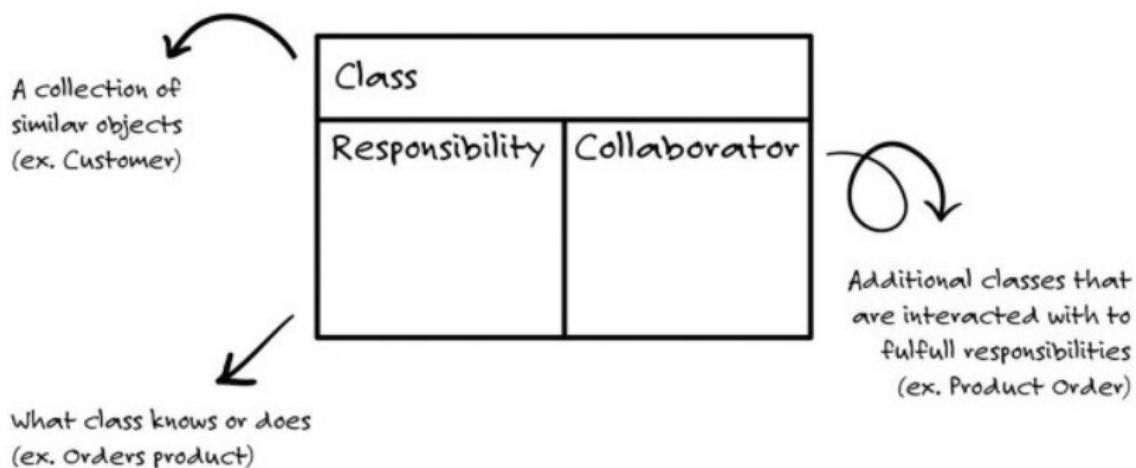


Polymorphism in OOD



Class-Responsability-Collaboration Cards (CRC)

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



Outline

- 1 Software Development
- 2 Software Architecture
- 3 Object-Oriented Design
- 4 Domain-Driven Design
- 5 Design Patterns



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.

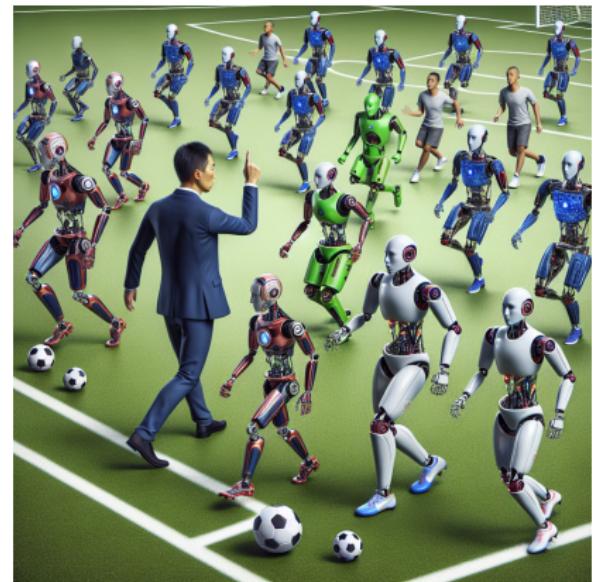


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



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.

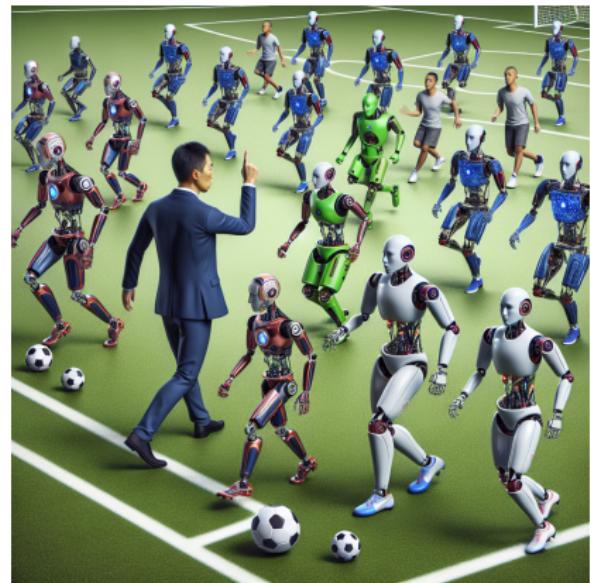


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



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

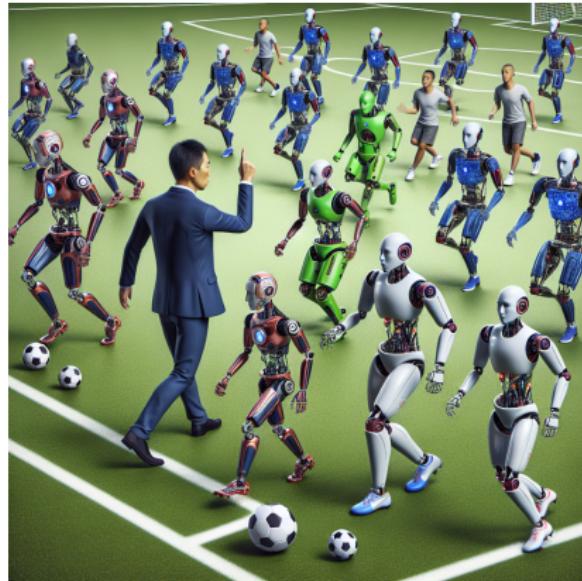


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



- 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:



Basics of Domain-Driven Design II

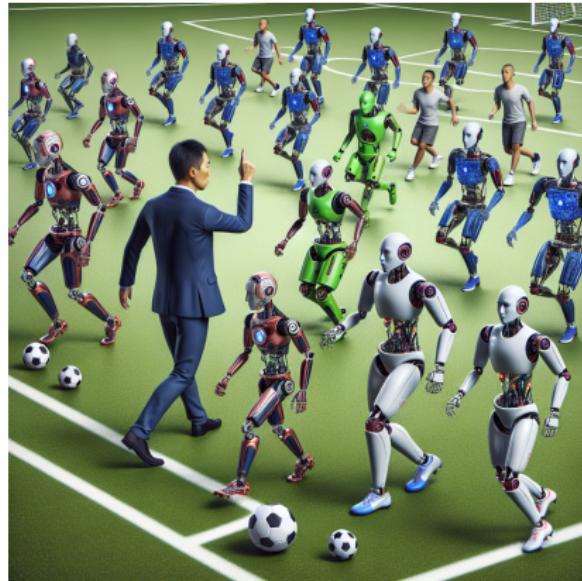


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



- 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:



Basics of Domain-Driven Design II

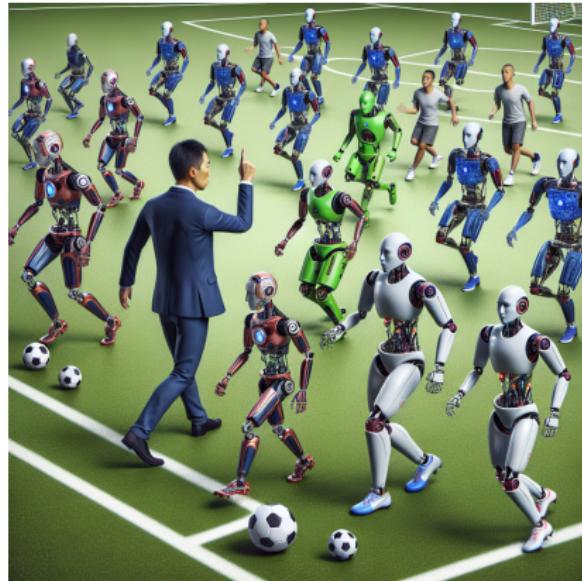


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



- 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

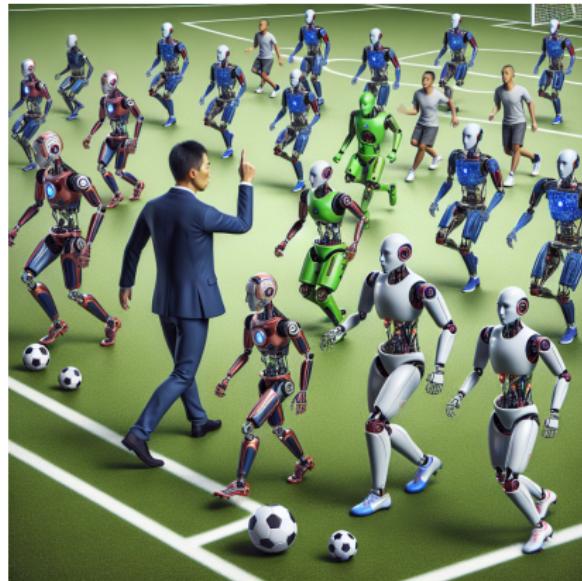


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



- 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

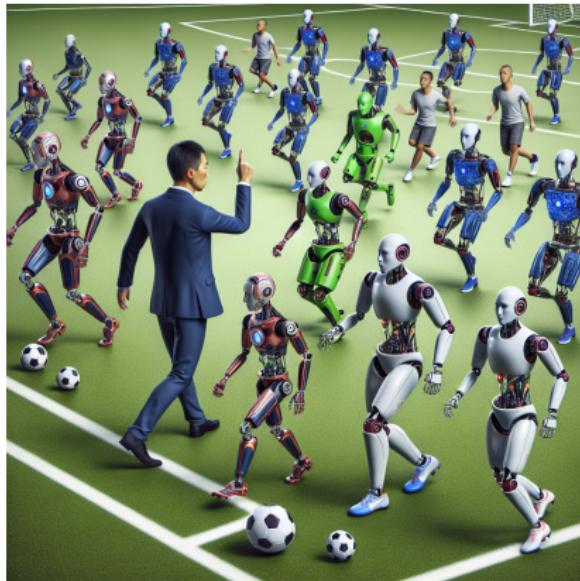


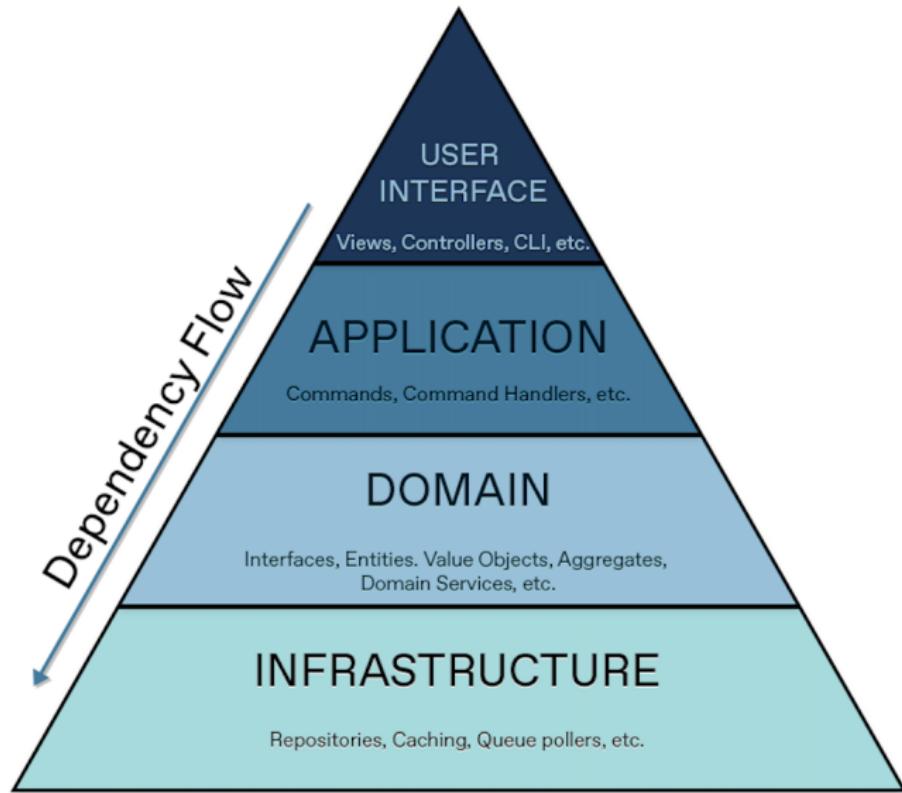
Figure: Prompt: Draw a soccer coach teaching robots soccer players.



- 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.



Business Logic in Layers



Outline

- 1 Software Development
- 2 Software Architecture
- 3 Object-Oriented Design
- 4 Domain-Driven Design
- 5 Design Patterns



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-tak-toe*.

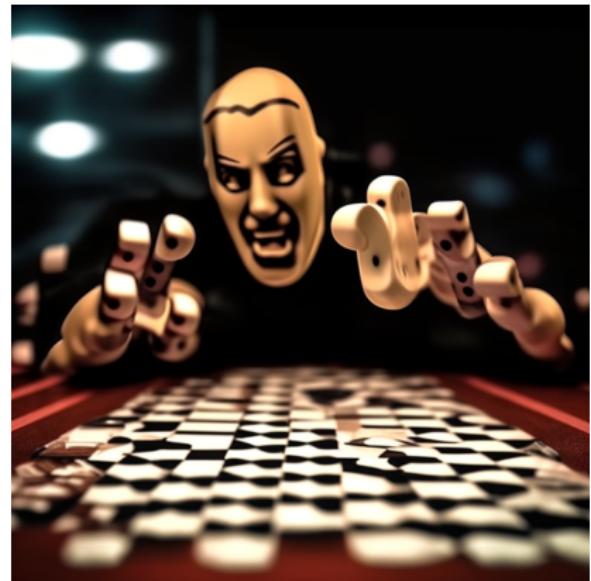


Figure: Prompt: Draw a tik-tak-inning strategy.



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-tak-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-tak-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**.



Basics of Design Pattern II



- 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**.

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



Outline

- 1 Software Development
- 2 Software Architecture
- 3 Object-Oriented Design
- 4 Domain-Driven Design
- 5 Design Patterns



Thanks!

Questions?



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

