

SOFTWARE ENGINEERING 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 Software Development



2 Object-Oriented Design



3 Domain-Driven Design



4 Software Methodologies



5 Information Systems



Outline

1 Software Development

2 Object-Oriented Design

3 Domain-Driven Design

4 Software Methodologies

5 Information Systems



1, 2, 3, 4, 5

Hola mundo

e

saludos
1 hola
2 mundo
3 saludos



Basics of Software Development I

The ~~main idea~~ is to solve ~~real-world~~ problems using ~~software solutions~~ engineers.

- The ~~main idea~~ is to solve ~~real-world~~ problems using ~~software solutions~~. One of the ~~main challenges~~ is the ~~complexity of systems~~, and learning how to manage it.

- It is not just about writing code; you must keep the entire software life cycle in mind. This means thinking about *design, testing, deployment, maintenance*, and many other tasks.



Figure: Prompt: Draw a python developer.



Basics of Software Development I

- The **main idea** is to **solve real-world** problems using **software solutions**. One of the **main challenges** is the *complexity of systems*, and learning how to **manage** it.
- It is **not just about writing code**; you must keep the entire **software life cycle** in mind. This means thinking about *design, testing, deployment, maintenance*, and many other tasks.

SDLC
B → G → D



Figure: Prompt: Draw a python developer.



Basics of Software Development II

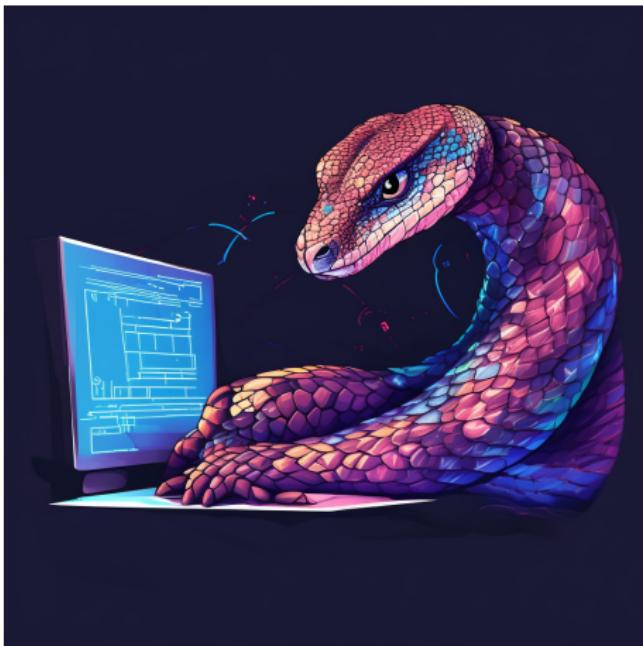


Figure: Prompt: Draw a python developer.



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

Basics of Software Development II



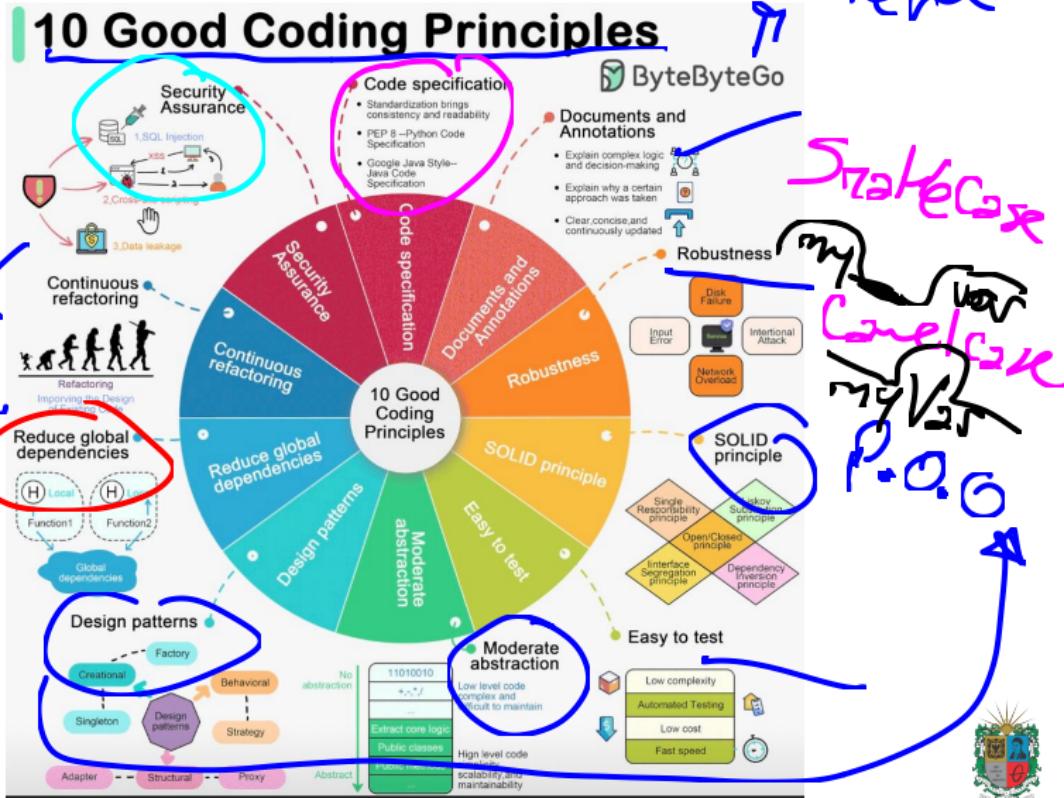
Figure: Prompt: Draw a python developer.



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

This is a crazy world.

10 Good Coding Principles



MSc. C.A. Sierra (UD FJC)

Software Engineering Seminar

2025-III

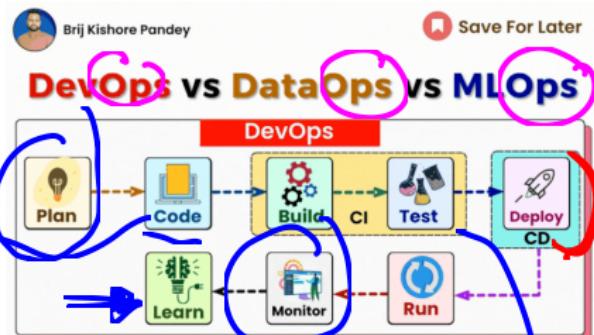
DataOps Vs. DevOps Vs. MLOps

MLOps

~
2 weeks
↔ client

CD

Continuous
Delivery /
Deploy most



DOps

Automable

CI

Continuous
Integration

No code



Basics of Software Architecture I

- It is important to **develop innovative** and **sophisticated software** to provide effective solutions for **end users' needs.** *? //*
- **Software architecture** brings innovation and a robust structure.
- The goal of software architecture is to minimize the human effort 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 effective solutions for end users' needs.
- **Software architecture** brings innovation and a robust structure.
- The **goal** of software **architecture** is to minimize the human effort 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**. It also helps to make 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**. It also helps to make 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**.



Types of Software Products

- **System Software:** Operating systems, device drivers, and utility programs.
- Application Software: Programs that perform specific user-oriented tasks (e.g. office suites, mobile apps).
- **2. Interoperability**: Software that connects disparate systems and facilitates communication.
- **3. Fast**: Specialized software designed to operate hardware in devices.
- Enterprise Software: Large-scale solutions like ERP, CRM, or SCM systems supporting business operations.



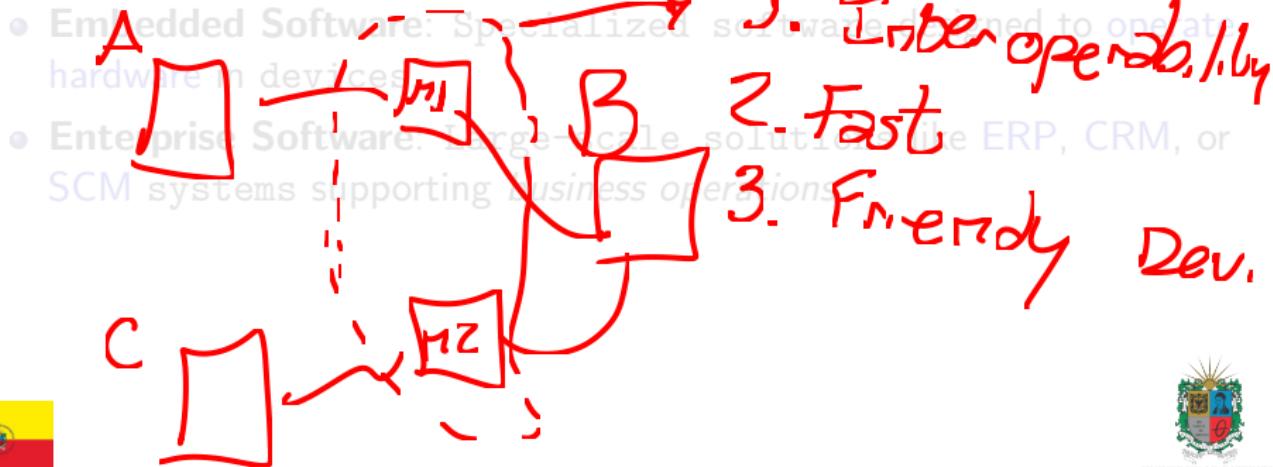
Types of Software Products

- **System Software:** Operating systems, device drivers, and utility programs.
 - **Application Software:** Programs that perform specific user-oriented tasks (e.g., office suites, mobile apps).
 - **Middleware:** Software that connects disparate systems and facilitates communication.
- 1. Roles + Activities**
- 2. Processes**
- 3. Tools + Techniques**
- 4. Robust + Scalable**



Types of Software Products

- **System Software:** Operating systems, device drivers, and utility programs.
- **Application Software:** Programs that perform specific user-oriented tasks (e.g., office suites, mobile apps).
- **Middleware:** Software that connects disparate systems and facilitates communication.



Types of Software Products

- **System Software:** Operating systems, device drivers, and utility programs.
- **Application Software:** Programs that perform specific user-oriented tasks (e.g., office suites, mobile apps).
- **Middleware:** Software that connects disparate systems and facilitates communication.
- **Embedded Software:** Specialized software designed to operate hardware in devices.
- **Enterprise Software:** Large-scale solutions like ERP, CRM, or SCM systems supporting business operations.
J. Robust
 - 2. Depends on hardware
 - 3. Flyweight



Types of Software Products

- **System Software:** Operating systems, device drivers, and utility programs.
- **Application Software:** Programs that perform specific user-oriented tasks (e.g., office suites, mobile apps).
- **Middleware:** Software that connects disparate systems and facilitates communication.
- **Embedded Software:** Specialized software designed to operate hardware in devices.
- **Enterprise Software:** Large-scale solutions like ERP, CRM, or SCM systems supporting business operations.



Supply Chain Management

1. Real-time \Rightarrow scalability + ultra fast

2. Friendly



Outline

Cache
F-3 ms

② Object-Oriented Design

auto-scaling

③ Domain-Driven Design

④ Software Methodologies

⑤ Information Systems



function setAge(τ Age)
 τ if (τ Age < 0)
 τ raiseError
 τ else
 τ age = τ Age

function get()
 τ . validate grants
 τ return age;

Basics of Object-Oriented Design I

- **Object-oriented programming** 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*).

actions



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



Basics of Object-Oriented Design II

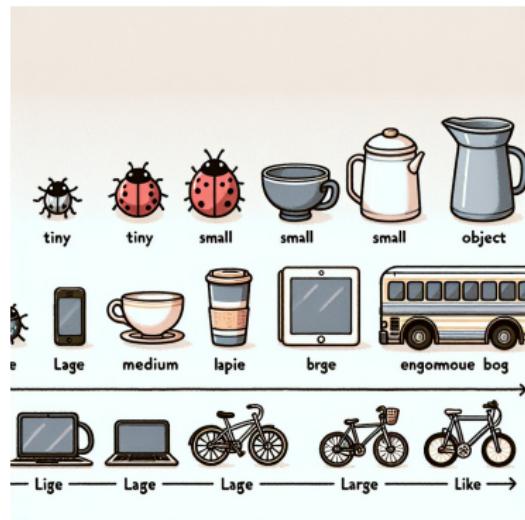


Figure: Prompt: Make an image of different real-world objects with binary inside each one.



- The idea is to design a **system** **modularly** to make it **easier** to **maintain** and **understand**. The idea is also to **emphasize** the **reuse of code**

The main principles of OOD

are:

Utilities
Package
import => P.O.O.

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** to make it **easier** to **maintain** and understand. The idea is also to **emphasize** the **reuse of code**.
- The **main principles** of **OOD** are:
 - Encapsulation
 - Abstraction
 - Inheritance
 - Polymorphism

hide
 in many ways
 transfer
 I
 multiple
 ways some
 action



Abstraction in OOD



sport

name

weight

height

~~age~~

medical_hist

sport

category

student

name

code

~~age~~

grades

program

birthday

email

worker

name

id

type_id

position

salary

Context



Encapsulation in OOD

```

class Student {
    private avg-grade double;
}

function isScholarship(min) {
    return avg_grade >= min;
}

function getAge() {
    year = current() - birthday.year
}

```

↓

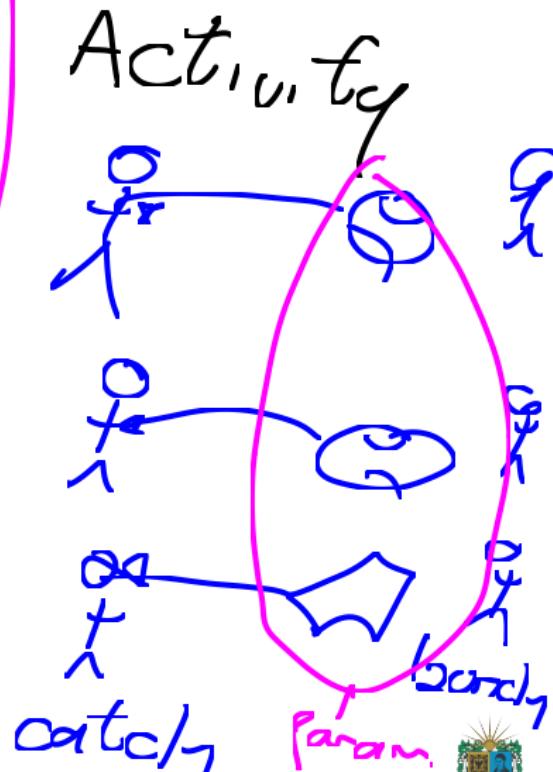
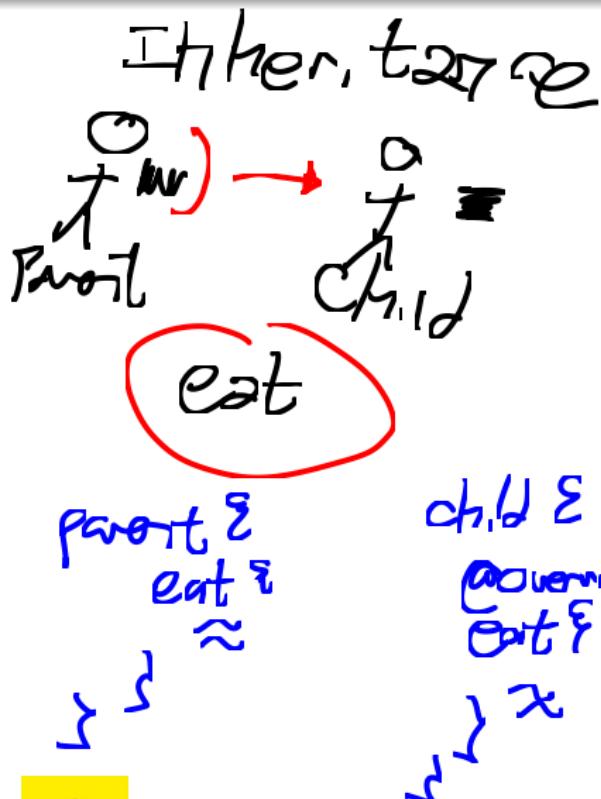
(IsEmpty)



Inheritance in OOD

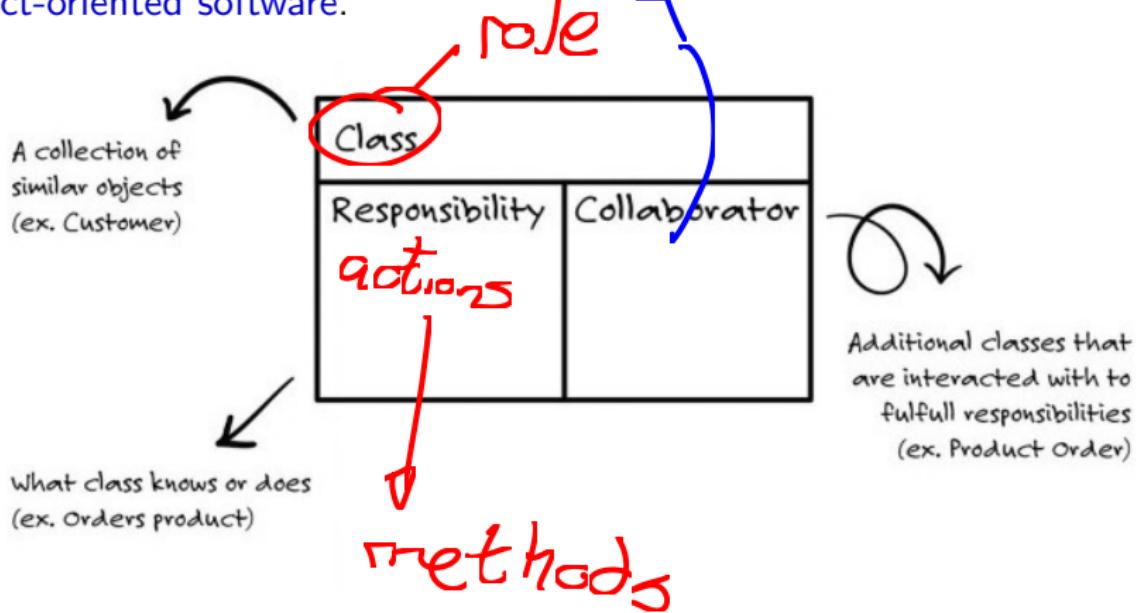


Polymorphism in OOD

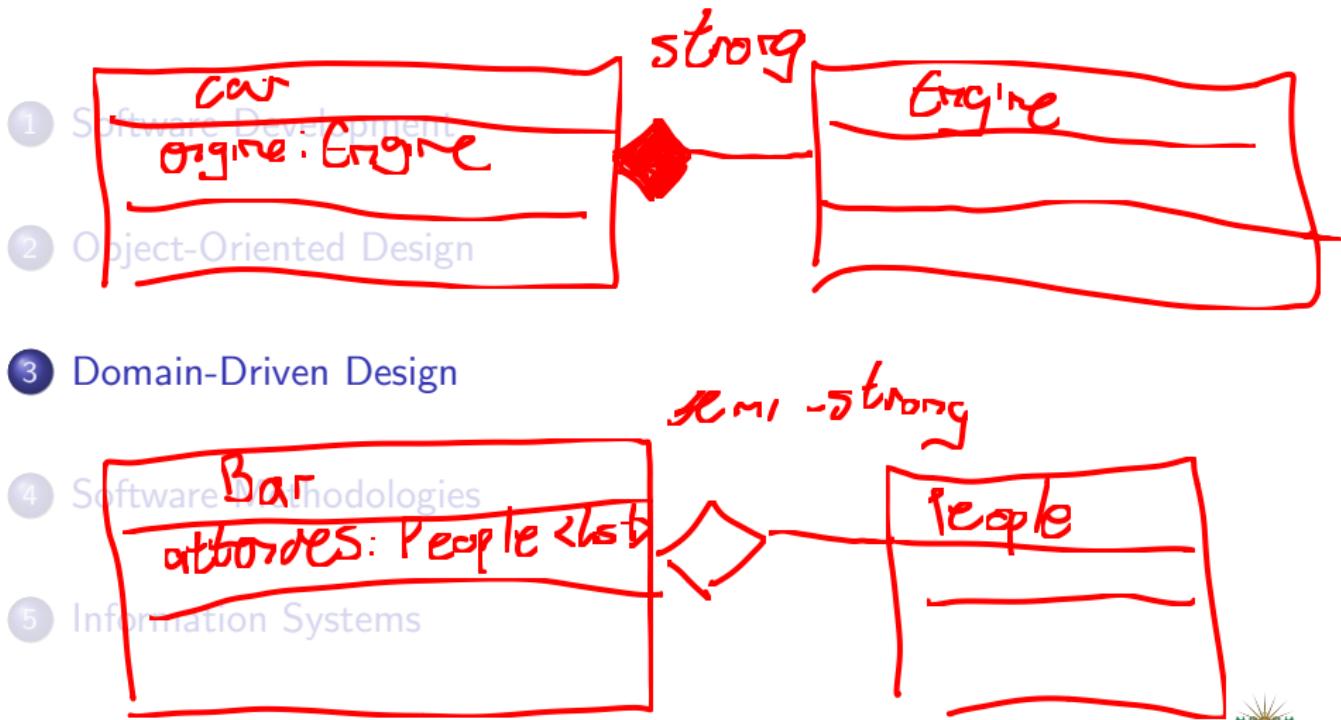


Class-Responsibility-Collaboration Cards (CRC)

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



Outline



Basics of Domain-Driven Design I

- DDD focuses 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 design a system to ensure alignment with real-world business needs.

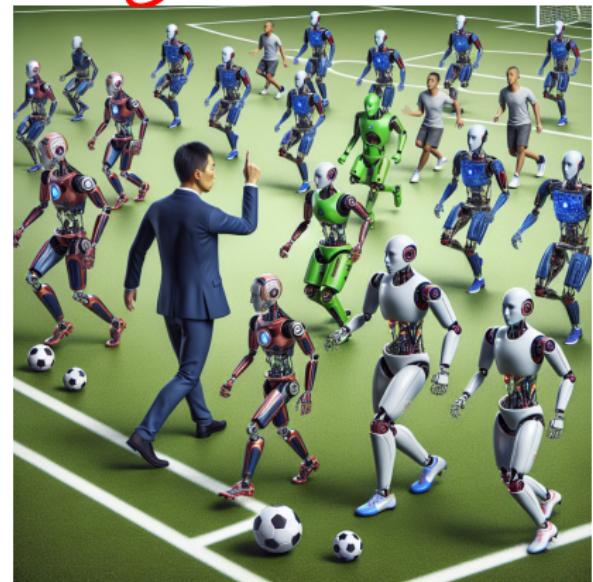


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



Basics of Domain-Driven Design I

- DDD focuses 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 **design** a **system** to ensure alignment with **real-world business needs**.

Non-functional requirements are based on the core

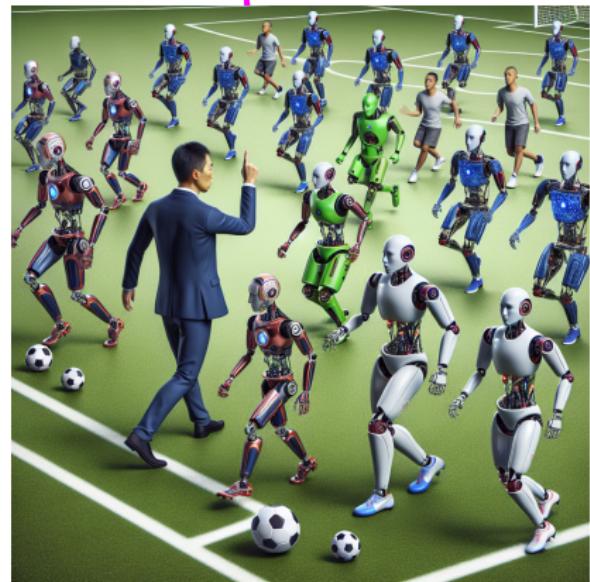


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



Basics of Domain-Driven Design II

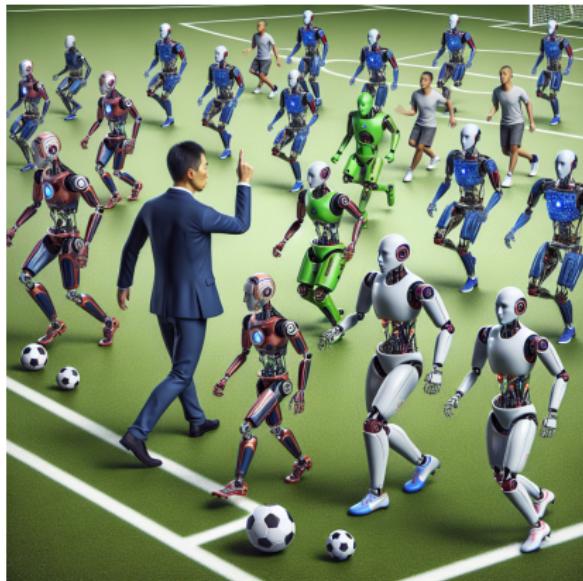


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



- The main principles of DDD are:

- Focus on the core domain.

- Base complex designs on models of the domain.

- Constantly collaborate with domain experts.

B.P.M.N.

Sequence
+ Activity
+ Process

Basics of Domain-Driven Design II

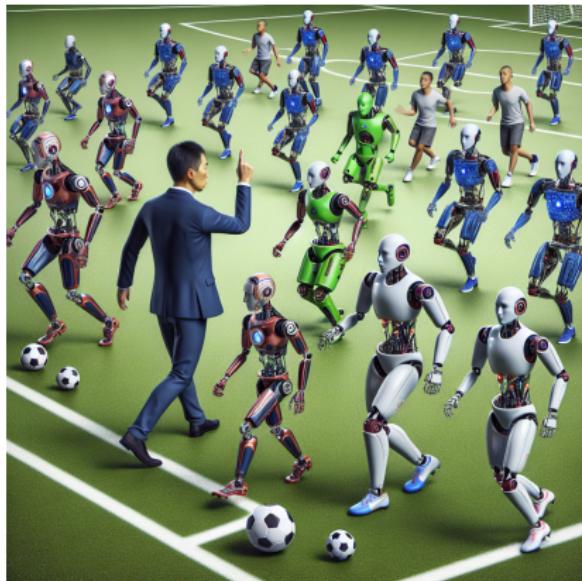


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



- 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 shown as follows:

Basics of Domain-Driven Design II

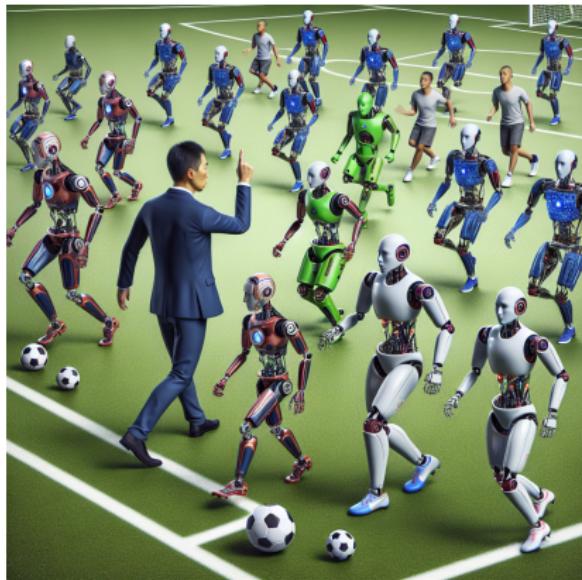


Figure: Prompt: Draw a soccer coach teaching robots soccer players.



- 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 shown as follows:
 - Domain Layer
 - Application Layer
 - Presentation Layer
 - Infrastructure Layer

Basics of Domain-Driven Design II

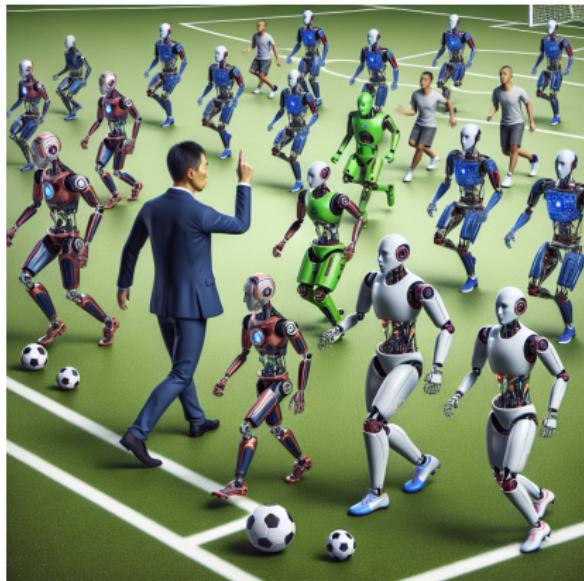


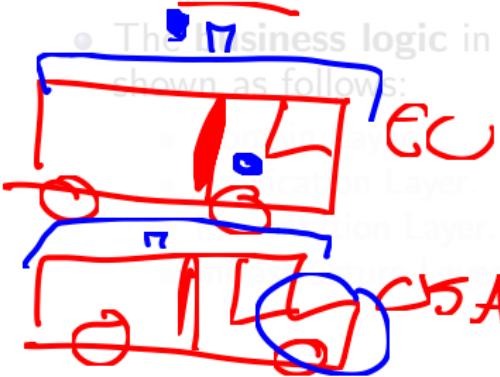
Figure: Prompt: Draw a soccer coach teaching robots soccer players.



- 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 shown as follows:



Basics of Domain-Driven Design II



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

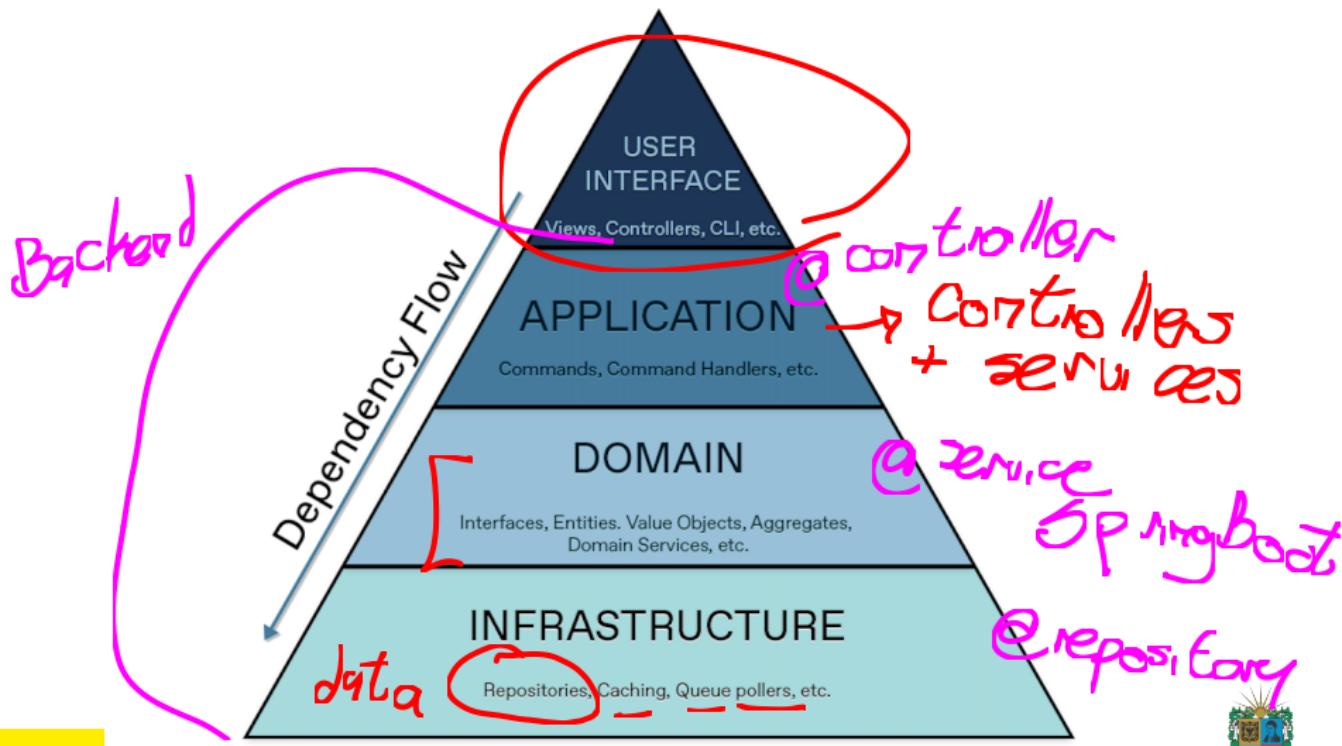


- The main principles of DDD are:
 - Focus on the core domain.
 - Base complex designs on models of the domain.
 - Constantly collaborate with domain experts.
 - Develop a knowledge-rich model.
- The business logic in layers is shown as follows:

- Domain Layer.
 - Application Layer.
 - Presentation Layer.
 - Infrastructure Layer.
- logic rules - sequence
data + objects*

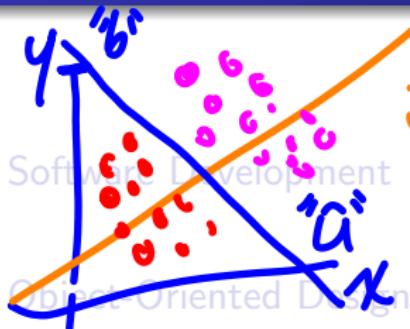


Business Logic in Layers

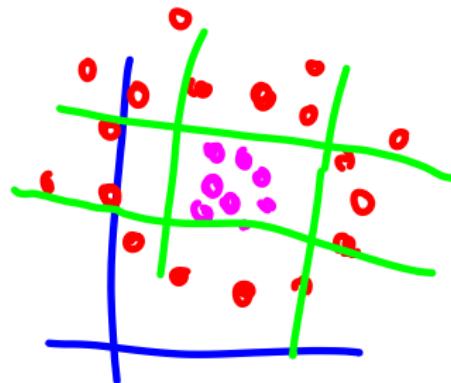
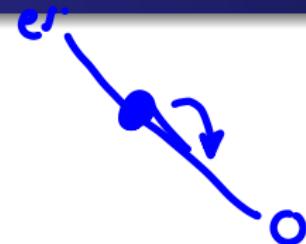


Outline

- 1 Software Development
- 2 Object-Oriented Design
- 3 Domain-Driven Design
- 4 Software Methodologies
- 5 Information Systems

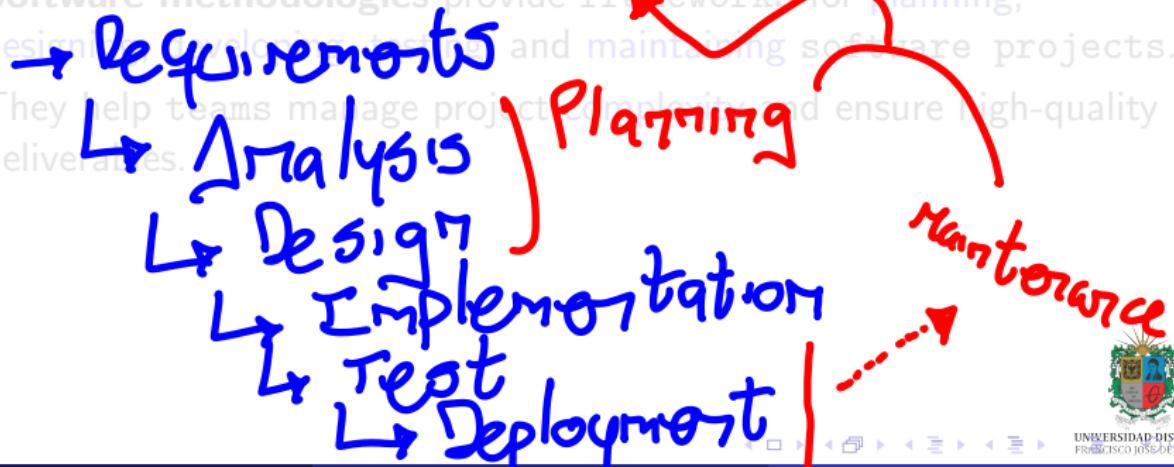


$$f = ax + b$$



Software Development Life Cycle (SDLC)

- The **SDLC** is a **framework** that describes the stages involved in developing software applications.
- It includes phases like **planning**, **analysis**, **design**, **implementation**, **testing**, and **maintenance**. | *dried*
- The **SDLC** helps ensure that software is developed in a **structured** and **efficient** manner, leading to **high-quality** products.
- Software methodologies provide frameworks for planning, designing, developing, testing, and maintaining software projects.
- They help teams manage project timelines and ensure high-quality deliverables.



Software Development Life Cycle (SDLC)

- The **SDLC** is a **framework** that describes the stages involved in developing software applications.
- It includes phases like planning, analysis, design, implementation, testing, and maintenance.
- The SDLC helps ensure that software is developed in a structured and efficient manner, leading to high-quality products.
- **Software methodologies** provide **frameworks** for **planning**, **designing**, **developing**, **testing**, and **maintaining** software projects.
- They help teams manage project **complexity** and ensure **high-quality** deliverables.

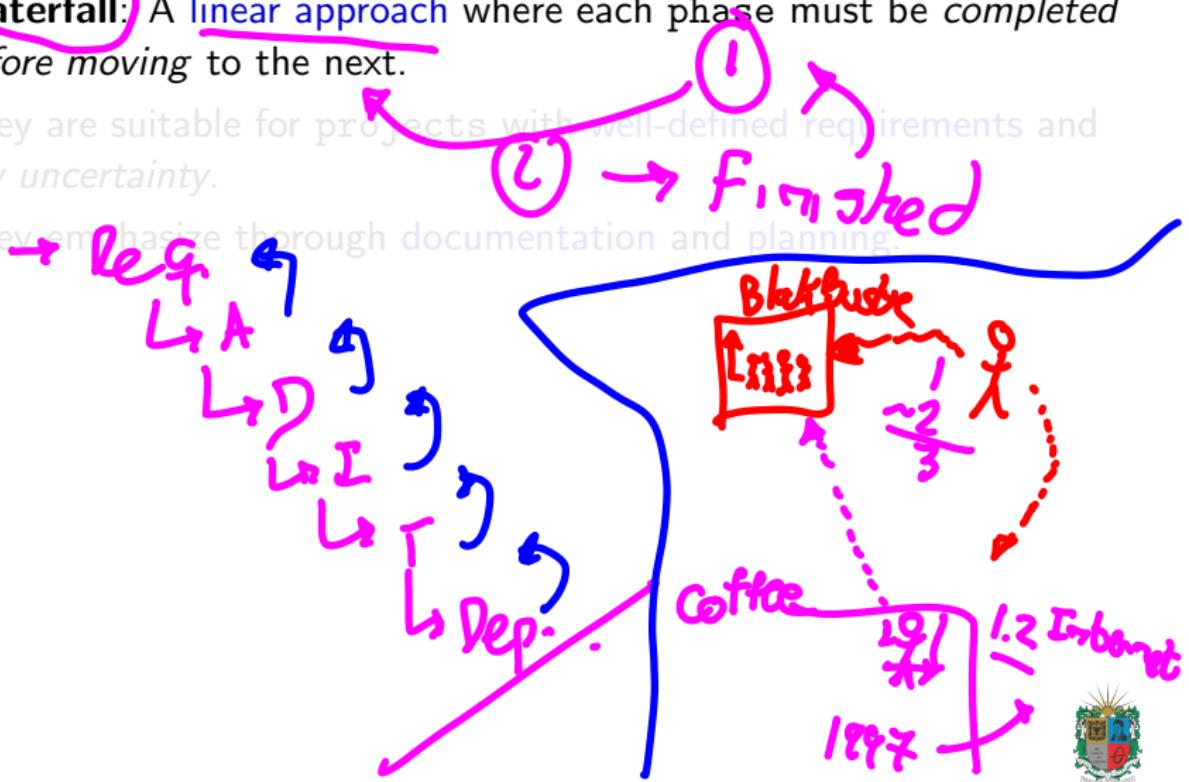
open
stochastic
#1 relations

at least 1
user → alive



Traditional Methodologies

- **Waterfall:** A linear approach where each phase must be *completed before moving* to the next.
 - They are suitable for projects with well-defined requirements and *low uncertainty*.
 - They emphasize thorough documentation and planning.

Traditional Methodologies

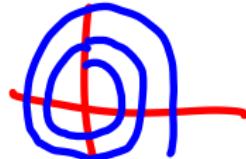
- **Waterfall:** A linear approach where each phase must be completed before moving to the next.
- They are suitable for projects with well-defined requirements and low uncertainty.
- They emphasize thorough documentation and planning.

60's

no books - tech
no internet

Government
colleges
top industry

MS-DOS
Macintosh } late 70's



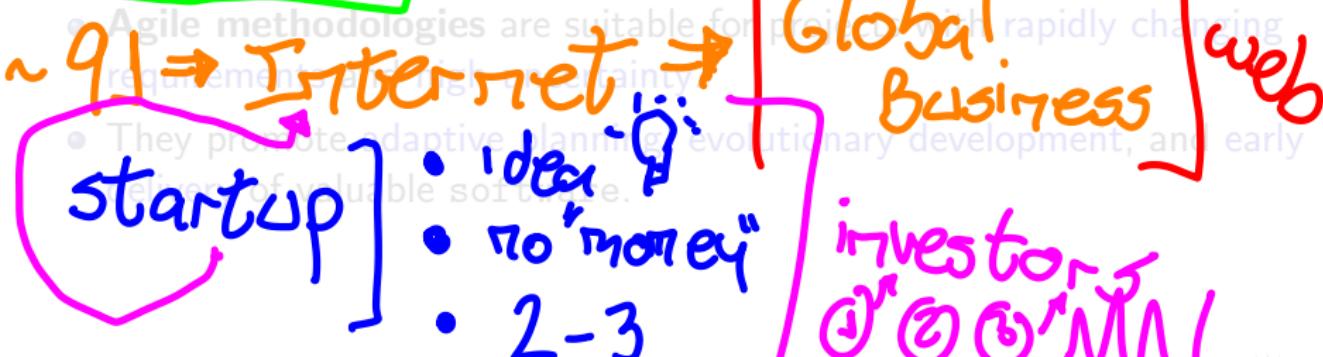
Agile Methodologies

- Emphasize iterative development, customer collaboration, and flexibility.
- They are based on the **Agile Manifesto**, which values individuals and interactions over processes and tools.
- Examples include Scrum, Kanban, Extreme Programming (XP), and Lean Software Development.

The Outliers

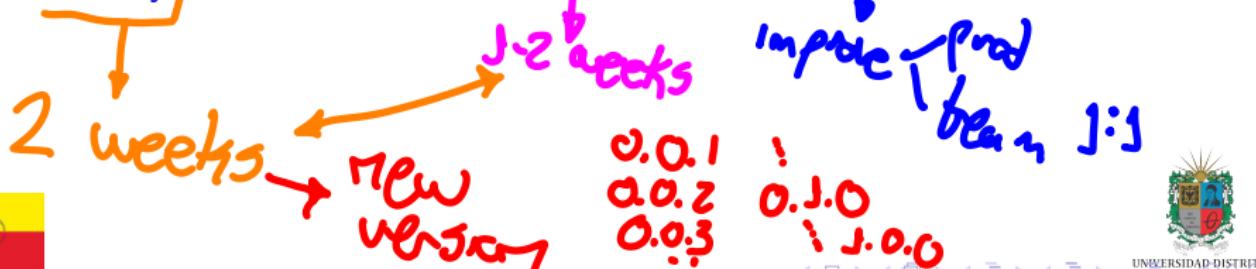
2001

team



Agile Methodologies

- Emphasize iterative development, customer collaboration, and flexibility.
- They are based on the **Agile Manifesto**, which values individuals and interactions over processes and tools.
- Examples include Scrum, Kanban, Extreme Programming (XP), and Lean Software Development.
- Agile methodologies are suitable for projects with rapidly changing requirements and high uncertainty.
- They promote adaptive planning, evolutionary development, and early delivery of valuable software.



Outline

1 Software Development

2 Object-Oriented Design

3 Domain-Driven Design

4 Software Methodologies

5 Information Systems



Information Systems

Data engineering

- An **Information System** is a system that collects, processes, stores, and disseminates information.
 - Information systems are used to support and manage business operations.
 - Examples of information systems include transaction processing systems, management information systems, decision support systems, executive information systems, expert systems, data warehouses, among others.
 - Information systems are used to automate and optimize business processes.
- **analysis**
- improve business intelligence**
- classification**
forecasting
data science



Information Systems

- An **Information System** is a **system** that *collects, processes, stores, and disseminates information*.
- **Information systems** are used to **support** and **manage** business operations.
- Examples of **information systems** include transaction processing systems, management information systems, decision support systems, executive information systems, expert systems, data systems, among others.
- **Information systems** are used to automate and optimize business processes.



Information Systems

- An **Information System** is a **system** that *collects, processes, stores, and disseminates information*.
- **Information systems** are used to **support** and **manage** business operations.
- Examples of **information systems** include transaction processing systems, management information systems, decision support systems, executive information systems, expert systems, data systems, among others.
- **Information systems** are used to **automate** and **optimize** **business processes**.

data source
machine learning
artificial agents



Data Systems

- A **Data System** is a **system** that collects, processes, stores, and retrieves **data**.
- Examples of data systems include **databases**, **data warehouses**, **data lakes**, **data marts**, **data cubes**, **data streams**, among others.
- Data systems** are used to **store** and **analyze** data.

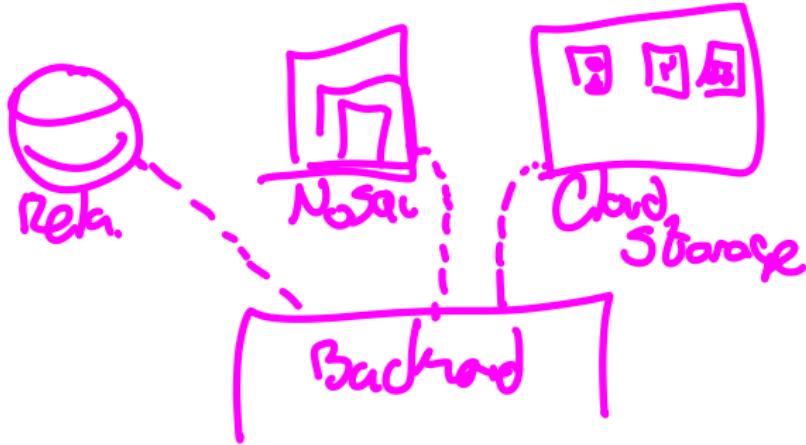
persistence

system

db ecosystem

Processed

raw



Expert Systems

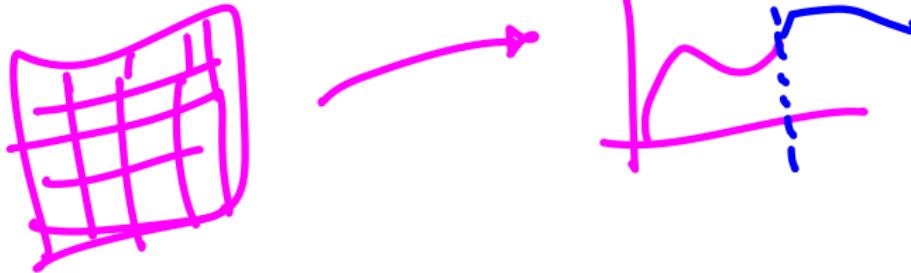
- An **Expert System** is a system that uses **knowledge** and **reasoning** to solve problems.
- Examples of expert systems include [diagnostic] systems, [predictive] systems, [prescriptive] systems, [decision support] systems, among others.
- Expert systems are used to automate and optimize decision-making processes.

regression,



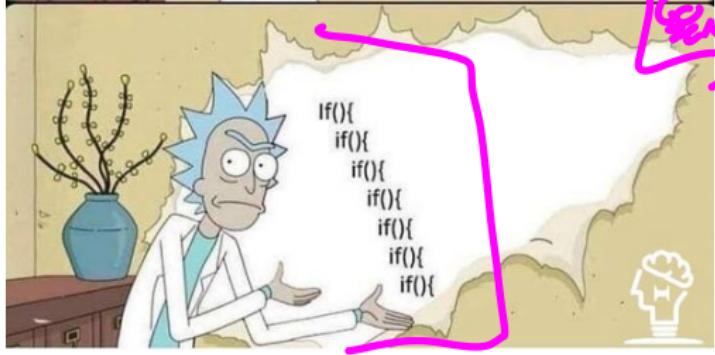
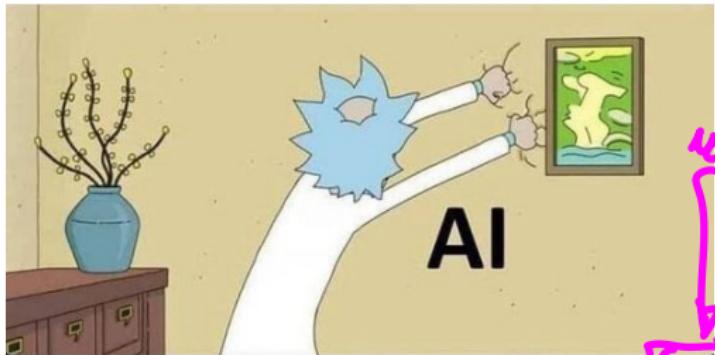
Expert Systems

- An **Expert System** is a system that uses **knowledge** and **reasoning** to solve problems.
- Examples of **expert systems** include diagnostic systems, predictive systems, prescriptive systems, decision support systems, among others.
- **Expert systems** are used to **automate** and **optimize** **decision-making processes**.



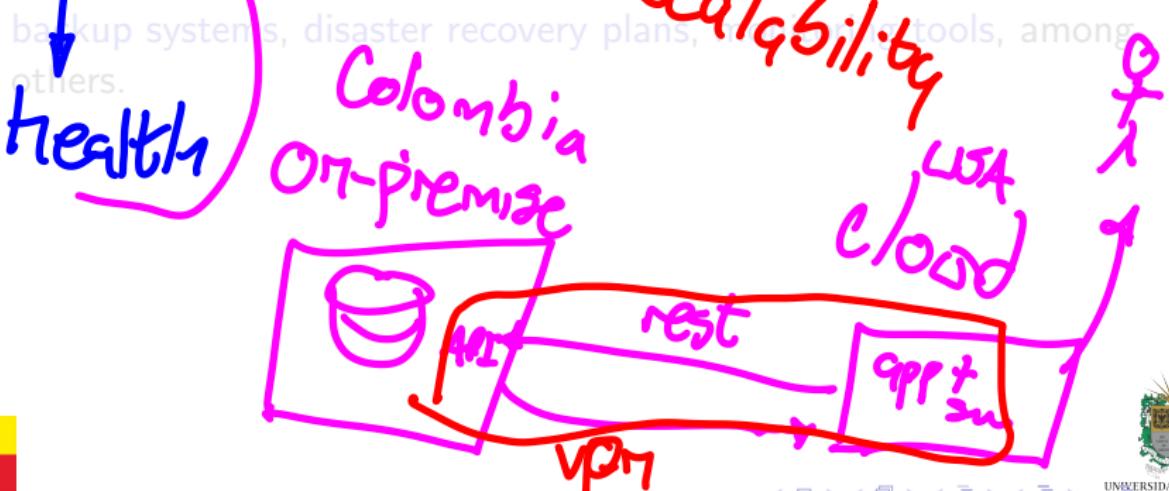
Expert Systems as Classical Artificial Intelligence

Here there is a great example of a diagnostic system.



Risks and Failures in Information

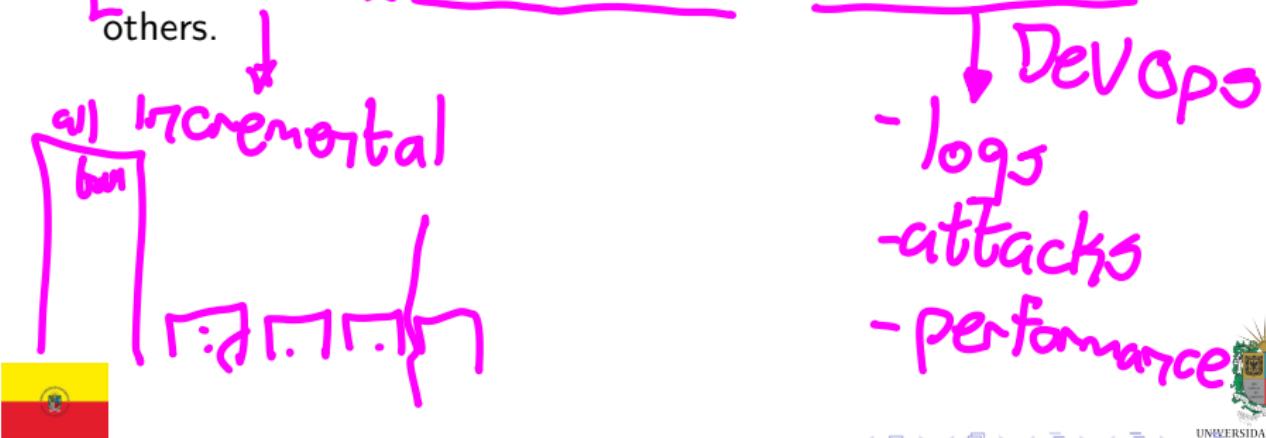
- Information systems are subject to **risks** and **failures** that can impact **business operations**.
- Examples of risks and failures include **security breaches**, **data loss**, **system downtime**, **performance issues**, **compliance violations** among others.
- Risks and failures can be mitigated through security measures, backup systems, disaster recovery plans, monitoring tools, among others.



Risks and Failures in Information

- **Information systems** are subject to **risks** and **failures** that can impact **business operations**.
- Examples of **risks and failures** include security breaches, data loss, system downtime, performance issues, compliance violations, among others.
- **Risks and failures** can be mitigated through **security measures**, **backup systems**, **disaster recovery plans**, **monitoring tools**, among others.

5



Outline

- 1 Software Development
- 2 Object-Oriented Design
- 3 Domain-Driven Design
- 4 Software Methodologies
- 5 Information Systems



Thanks!

Questions?



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

