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

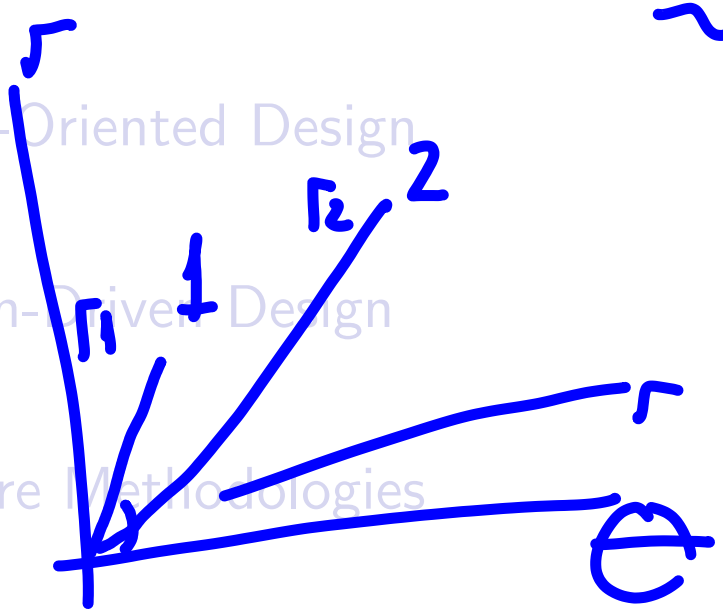
2025-III

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Outline



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.

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.

Figure: Prompt: Draw a python developer.

Basics of Software Development II

Figure: Prompt: Draw a python developer.

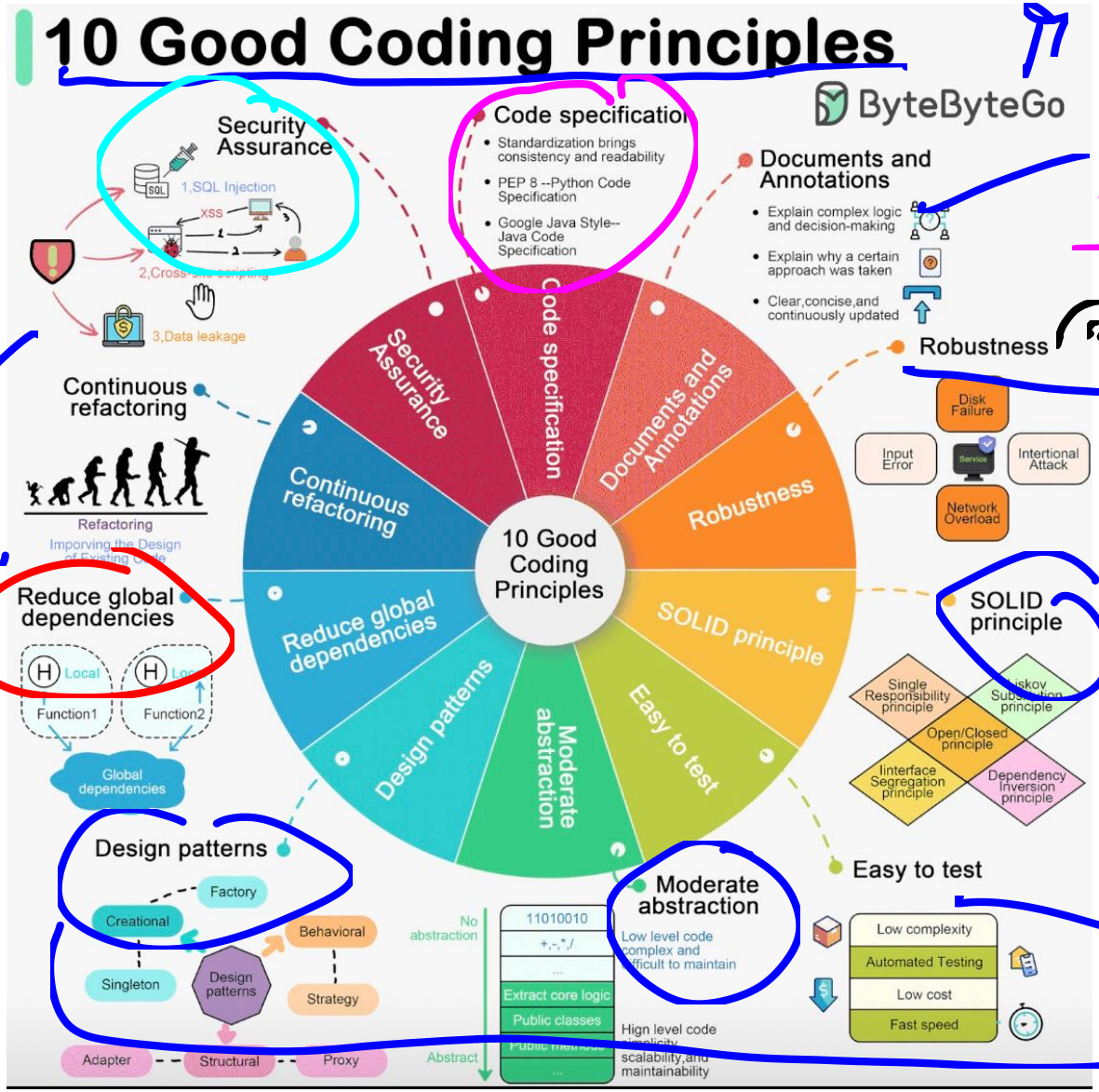
- However, **writing code** is the most important task, and it is the main skill to have. You can write code to automate tests, deployments, integrations, and more.
- It is also vital to know a lot about software design, to propose good solutions, and to **read every day** in order to choose and use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

Basics of Software Development II

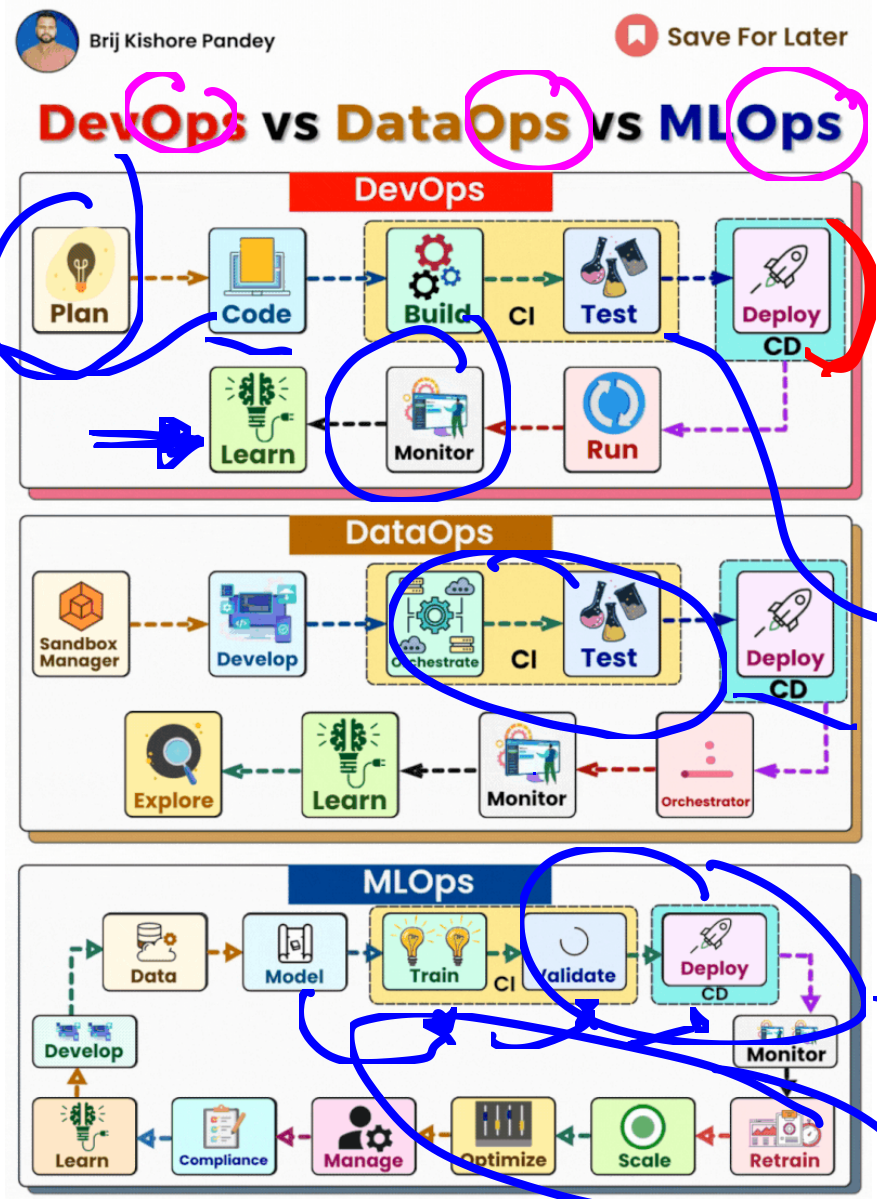
Figure: Prompt: Draw a python developer.

- 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



DataOps *Vs.* DevOps *Vs.* MLOps



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).
- **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*.

1. Tools → C
2. Interoperability
3. Fast

all

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Types of Software Products

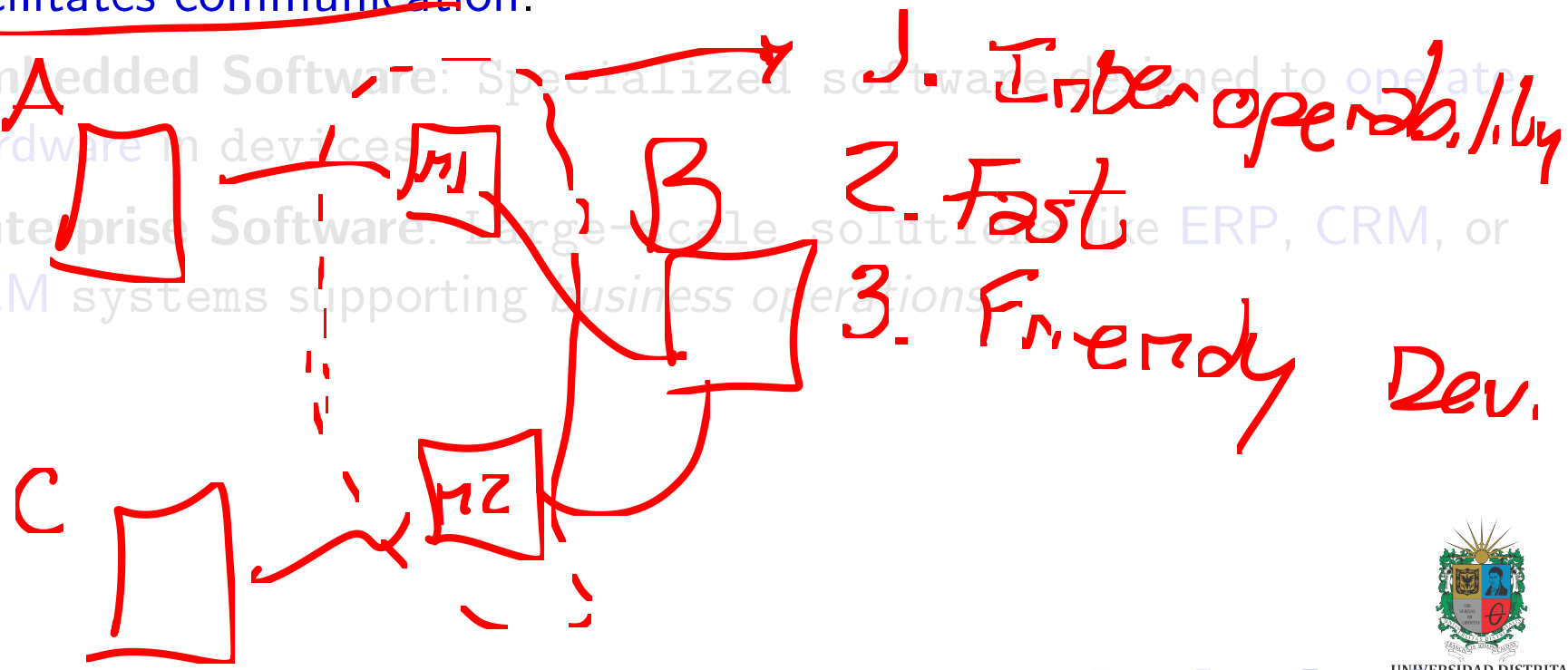
- **System Software**: Operating systems, device drivers, and utility programs.
- **Application Software**: Programs that perform specific user-oriented tasks (e.g., office suites, mobile apps).
- **Middleware**: Software that connects disparate systems and facilitates communication.
- **Embedded Software**: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

1. Roles + Activities
2. Processes
3. Tools + Techniques
4. Robust + Scalable

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

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.



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.

1. Robust
2. Depends on Hardware
3. Flyweight

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

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*.

Outline

Cache
1-3 ms

auto-scaling

function setAge(nAge) {
if (nAge < 0)
raiseError
else
age = nAge
}

function get() {
validate grants
return age;
}

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

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*).

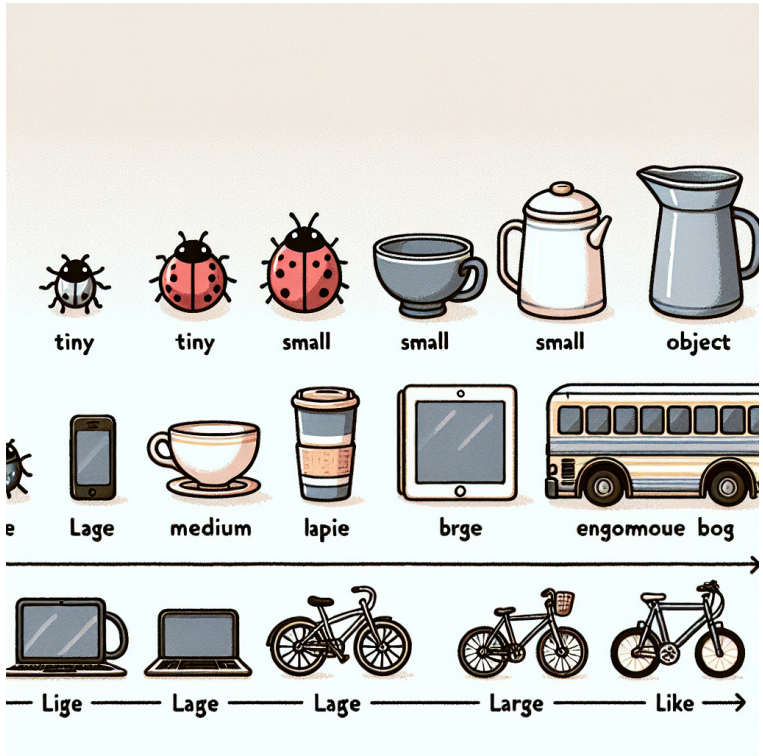


Figure: Prompt: Make an image of different real-world objects with binary inside each one.

Basics of Object-Oriented Design II

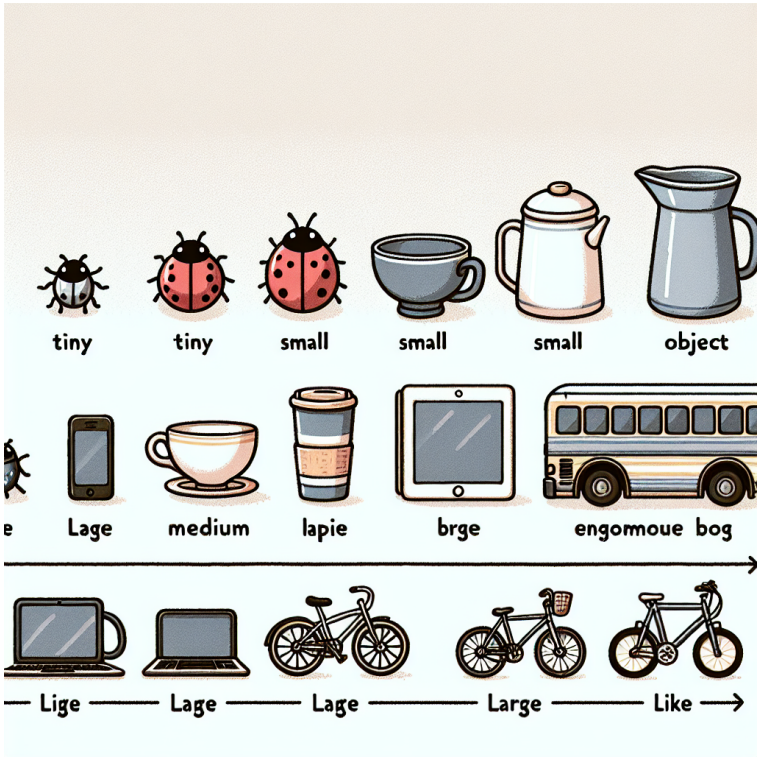


Figure: Prompt: Make an image of different real-world objects with binary inside each one.

- 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:
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

utilities

Package

import => P.O.O.

Basics of Object-Oriented Design II

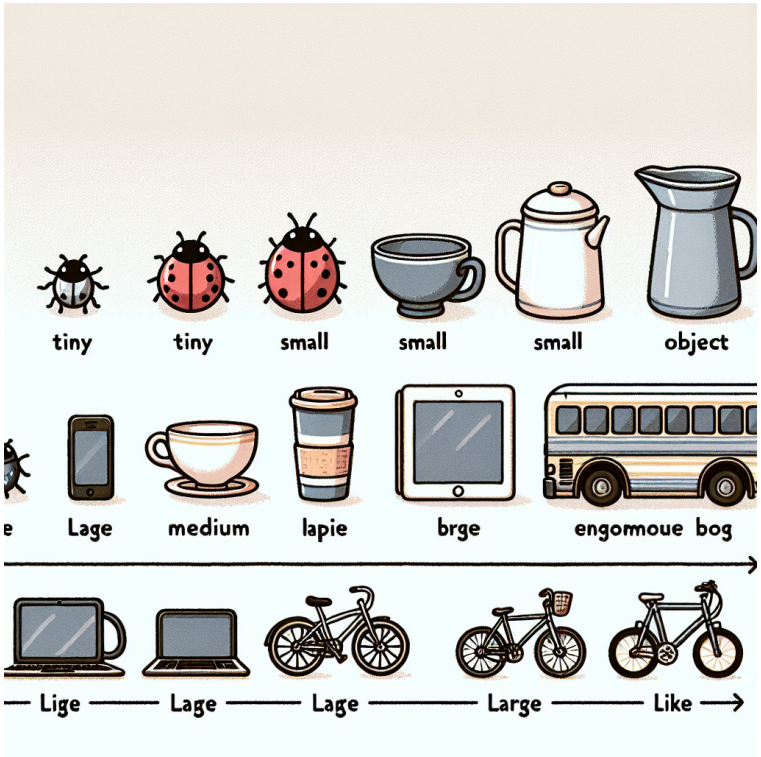
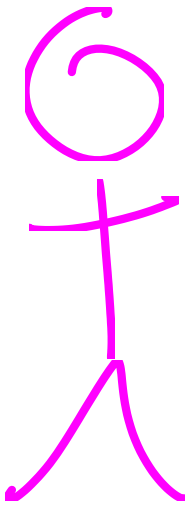


Figure: Prompt: Make an image of different real-world objects with binary inside each one.

- 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

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

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

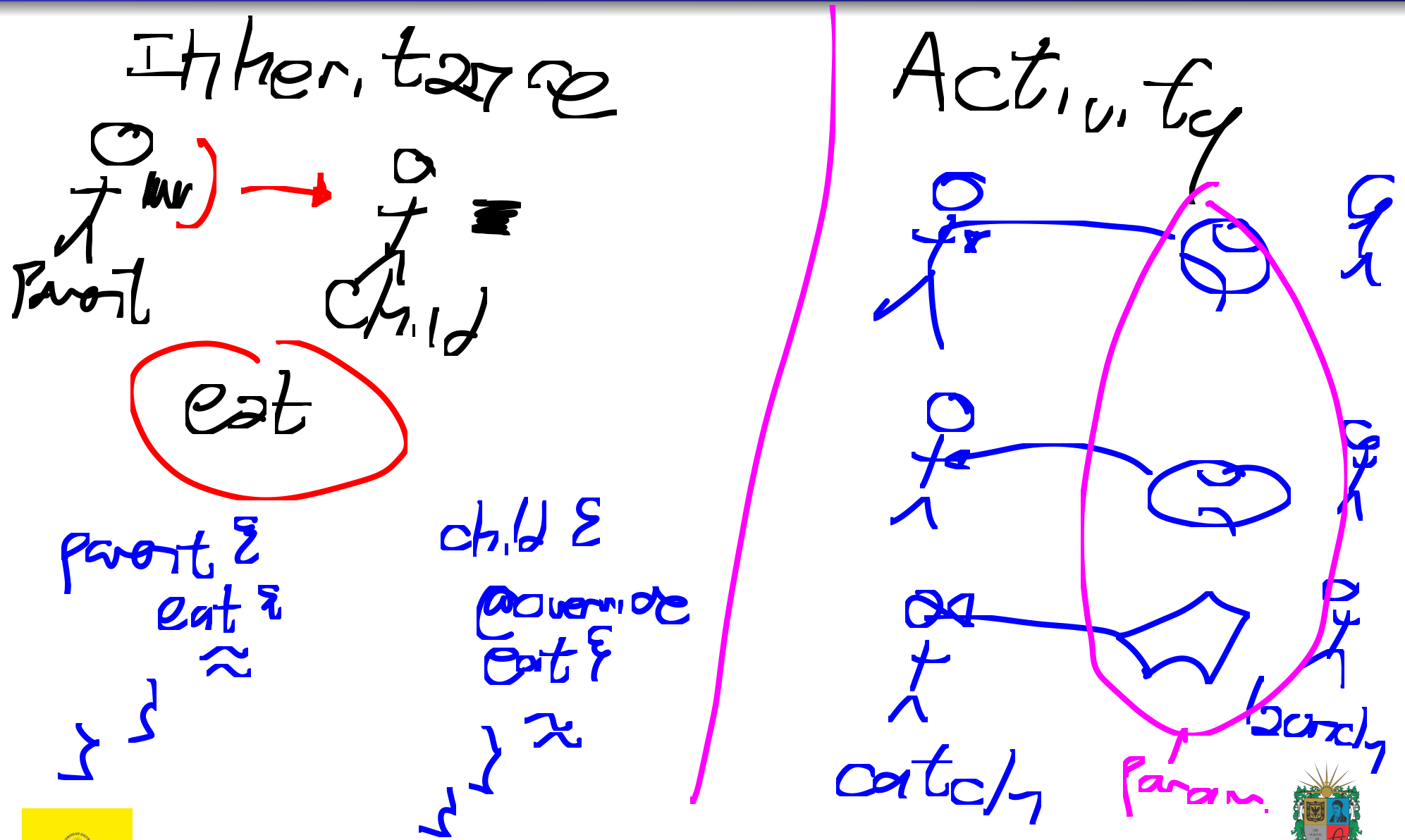
Encapsulation in OOD

```
class Student {
private avg-grade double;

function is-scholarship(min) {
    return avg_grade >= min ?
        true : false;
}

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

isEmpty

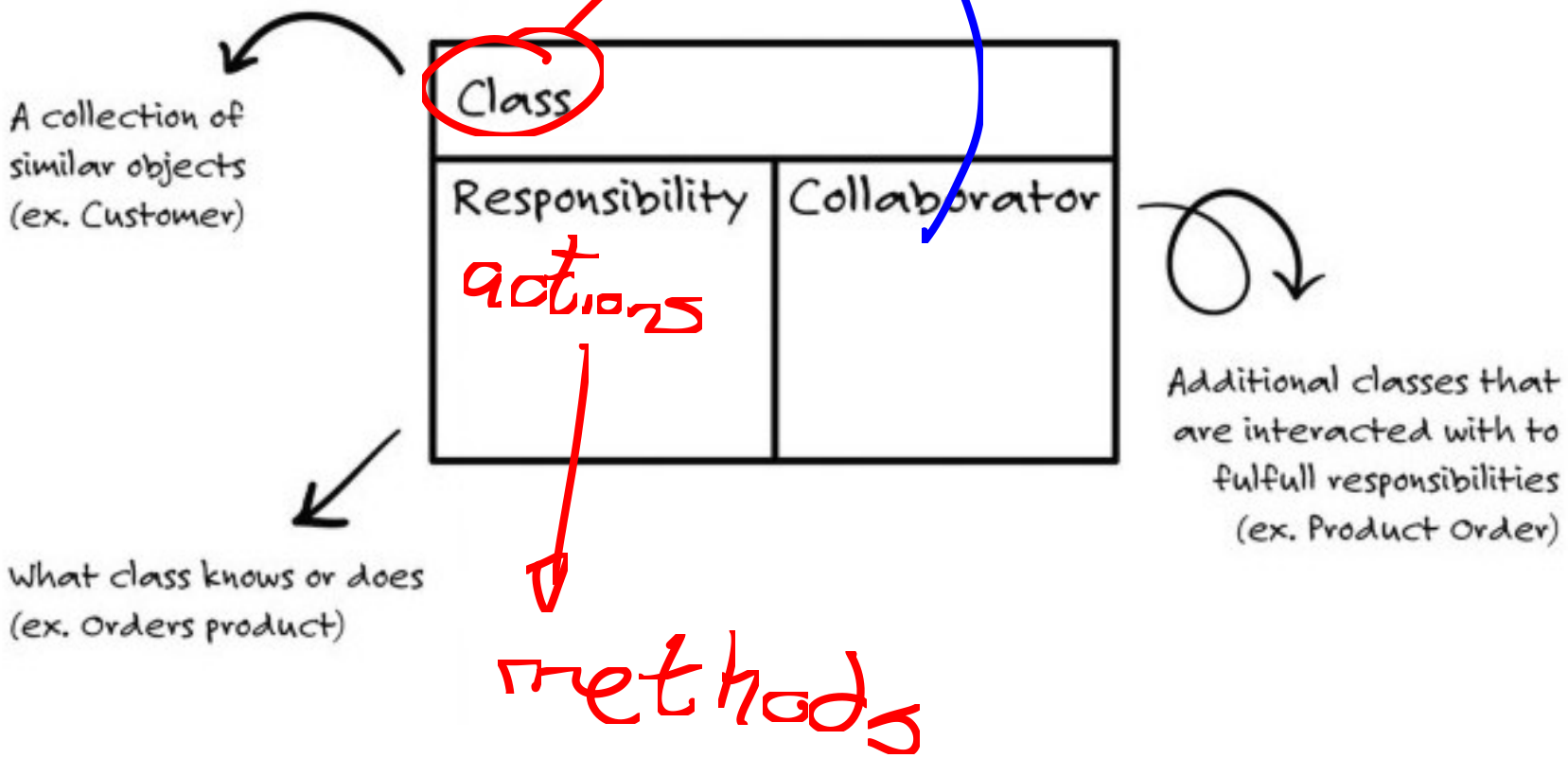
Inheritance in OOD

Polymorphism in OOD

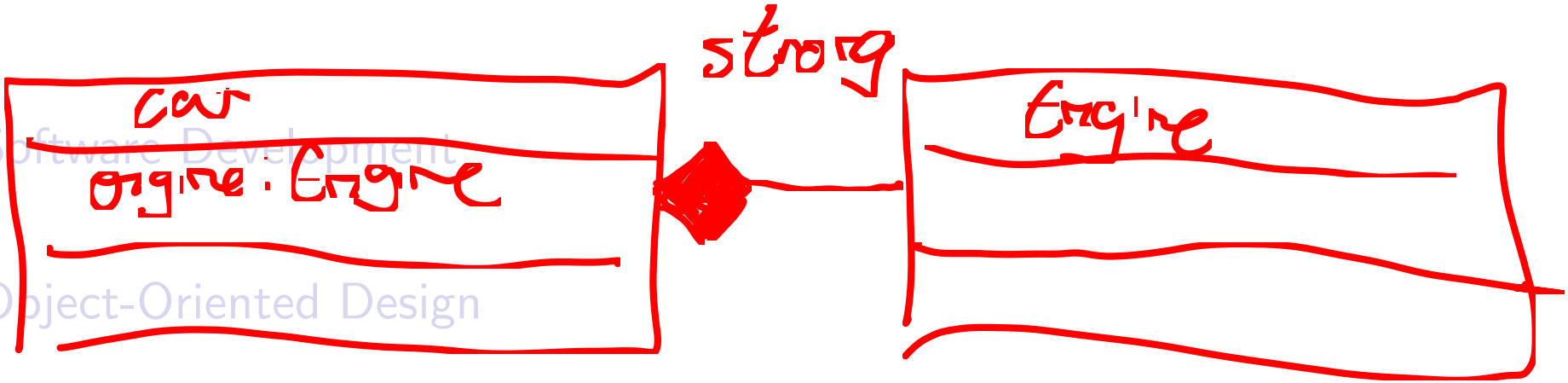


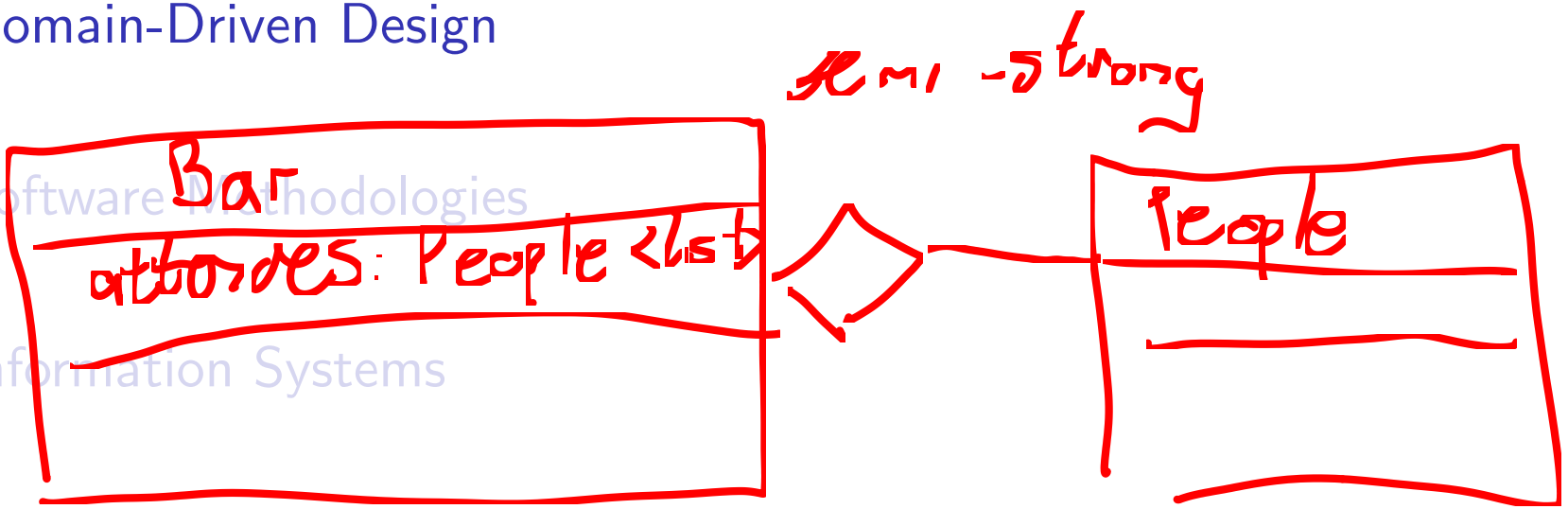
Class-Responsibility-Collaboration Cards (CRC)

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



Outline





UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

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.

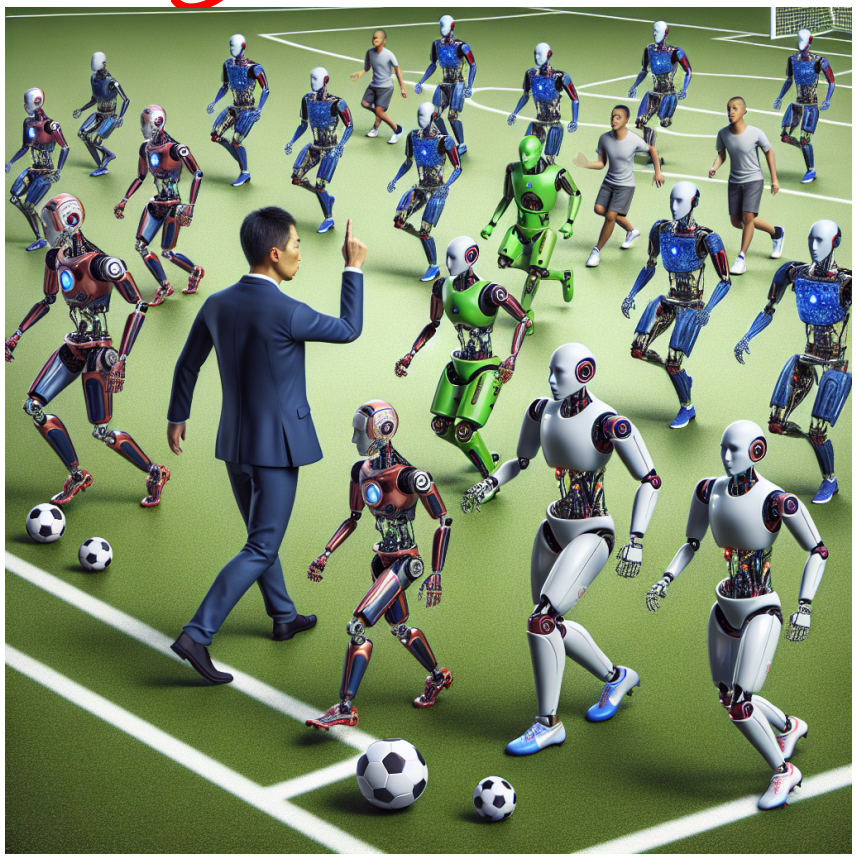


Figure: Prompt: Draw a soccer coach teaching robots soccer players.

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.

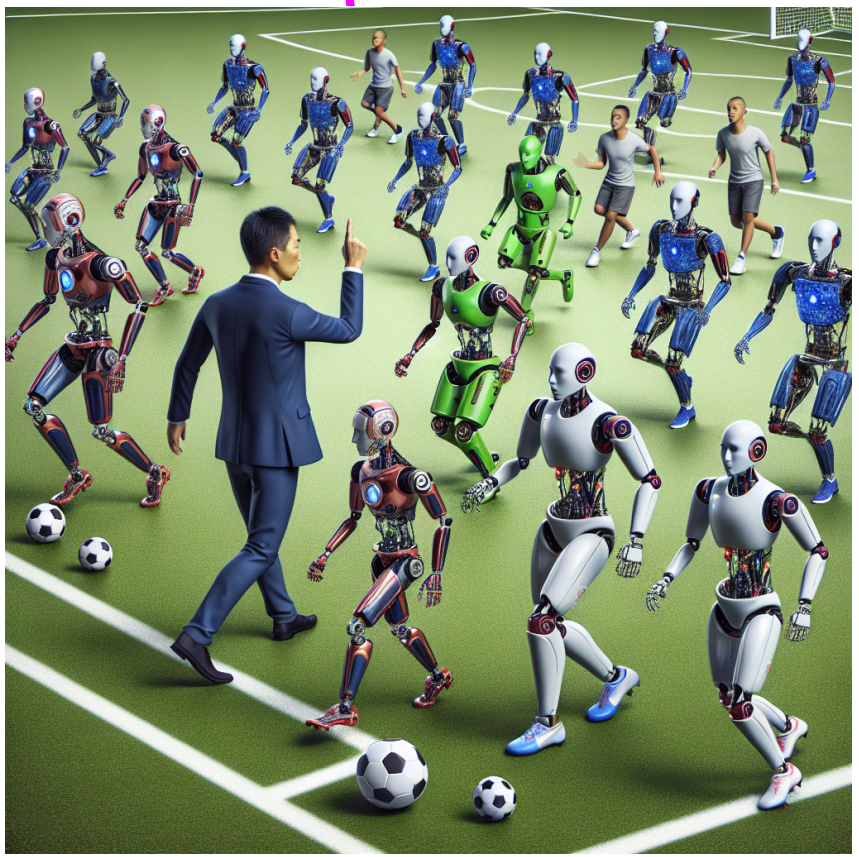


Figure: Prompt: Draw a soccer coach teaching robots soccer players.

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

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:
  - 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:
  - Domain Layer.
  - Application Layer.
  - Presentation Layer.
  - Infrastructure Layer.

Basics of Domain-Driven Design II

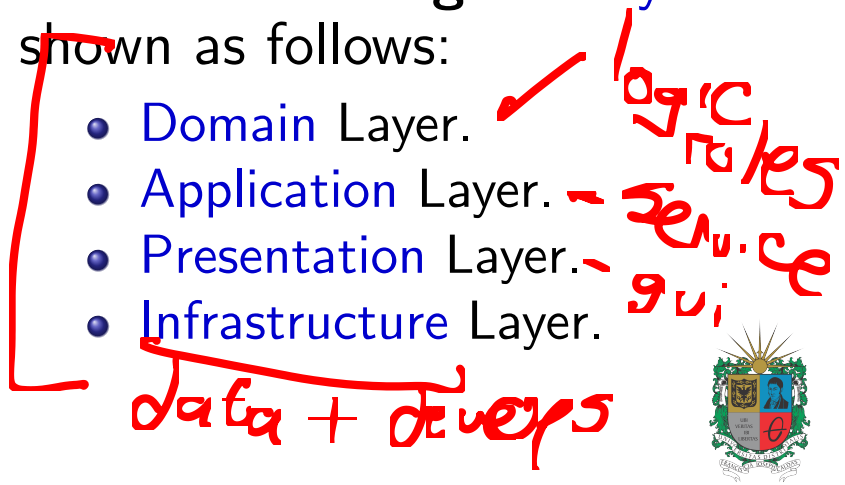
Figure: Prompt: Draw a soccer coach teaching robots soccer players.

- 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.

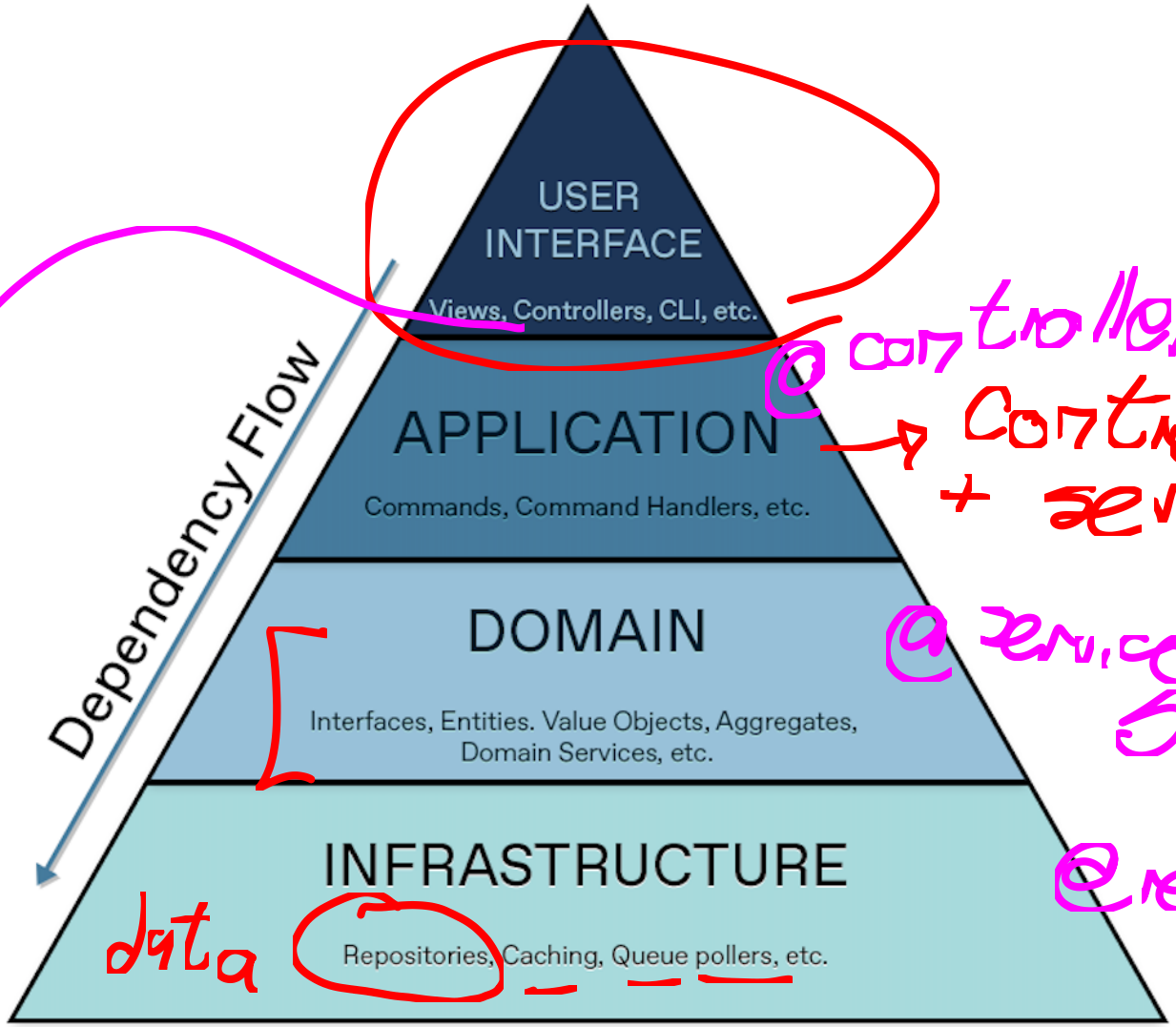
Basics of Domain-Driven Design II

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

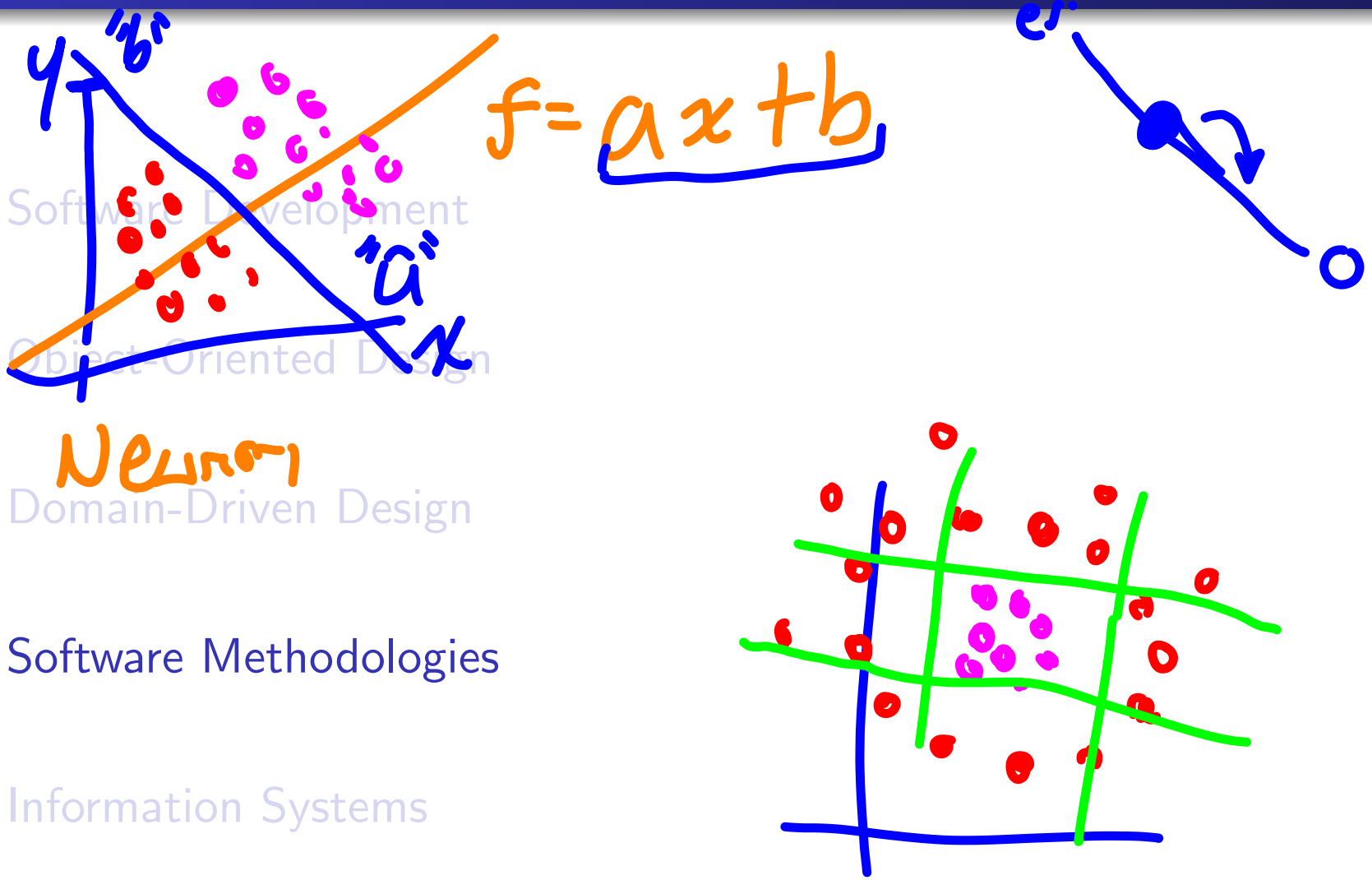
- The main principles of **DDD** are:
  - Focus on the core domain.
  - Base complex designs on models of the domain.
  - Constantly collaborate with domain experts.
  - Develop a knowledge-rich model.
- The **business logic** in layers is shown as follows:
  - Domain Layer.
  - Application Layer.
 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



Business Logic in Layers



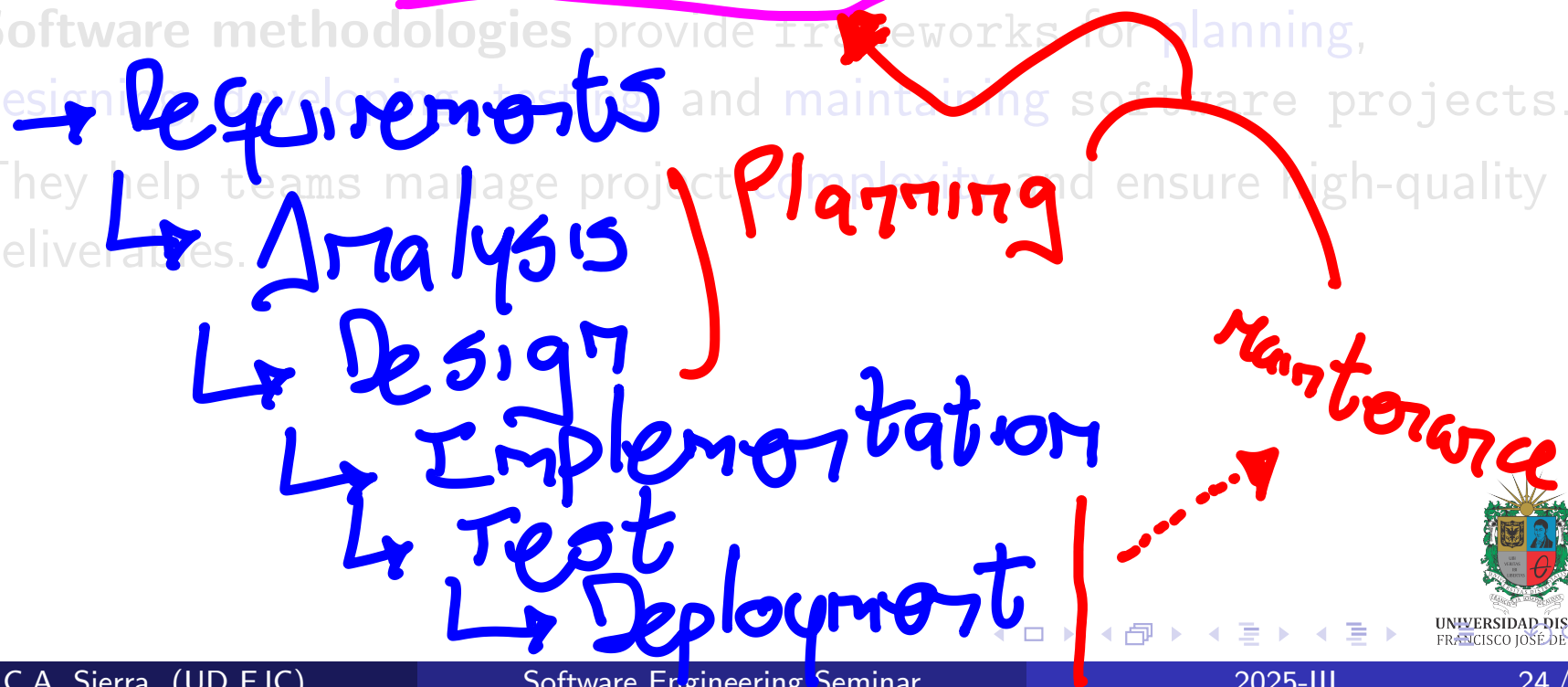
Outline



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

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.



Software Development Life Cycle (SDLC)

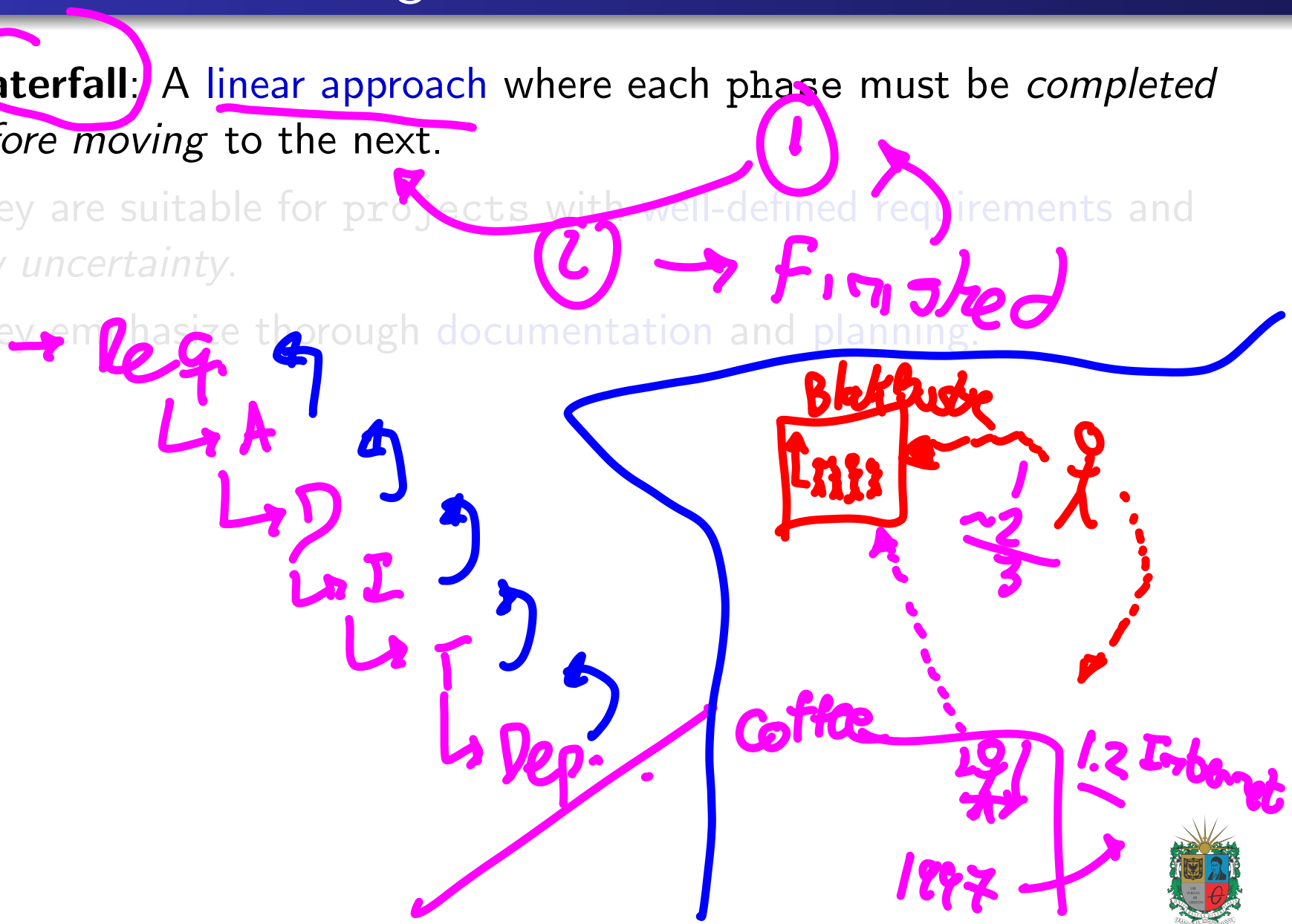
- The **SDLC** is a framework that describes the stages involved in developing software applications.
- It includes phases like `planning`, `analysis`, `design`, `implementation`, `testing`, and `maintenance`.
- The SDLC helps ensure that software is developed in a structured and efficient manner, leading 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

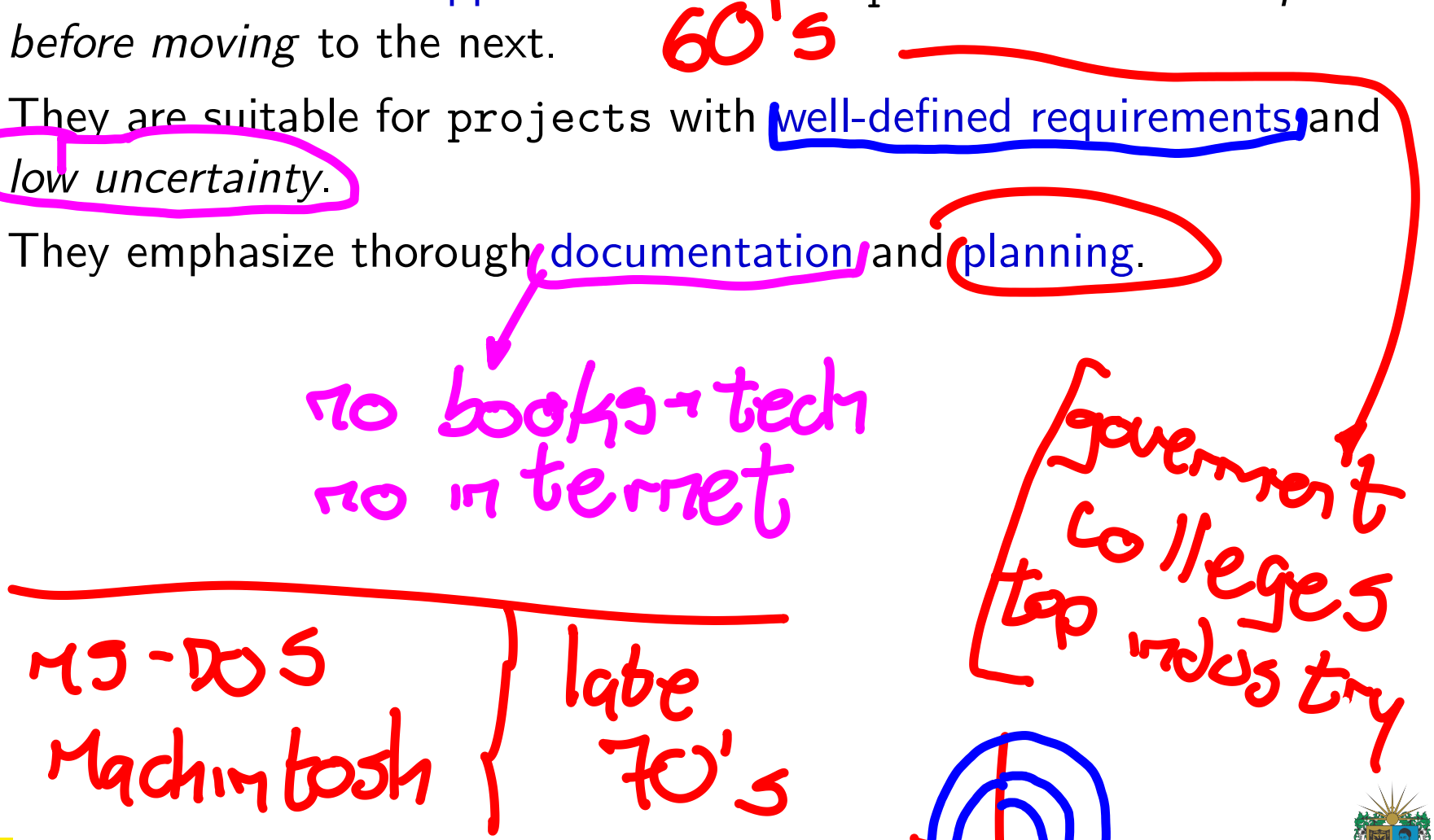
- **Waterfall**: A linear approach where each phase must be *completed before moving* to the next.
- They are suitable for projects with well-defined requirements and *low uncertainty*.
- They emphasize thorough documentation and planning.



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

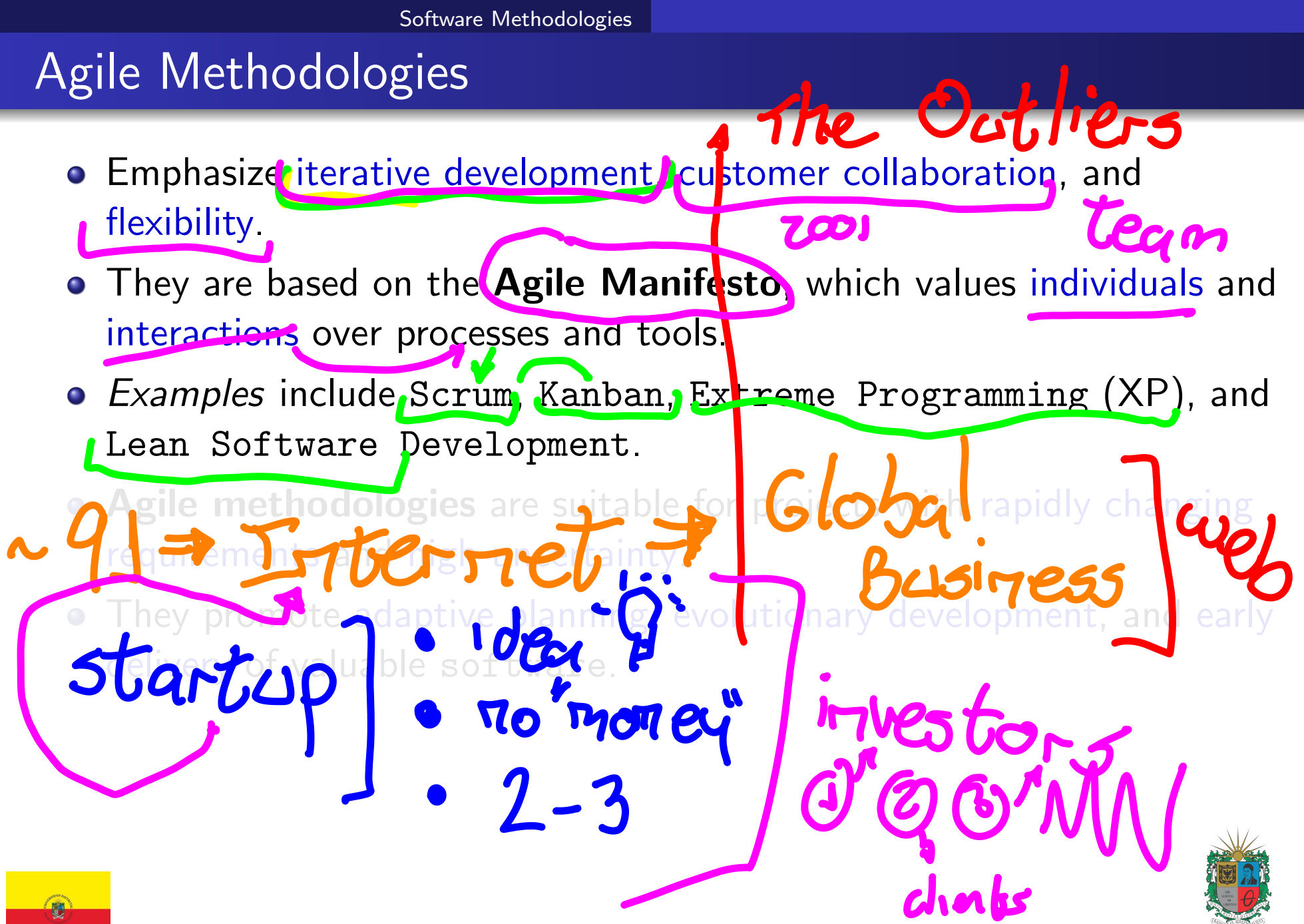
Traditional Methodologies

- **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.



Agile Methodologies

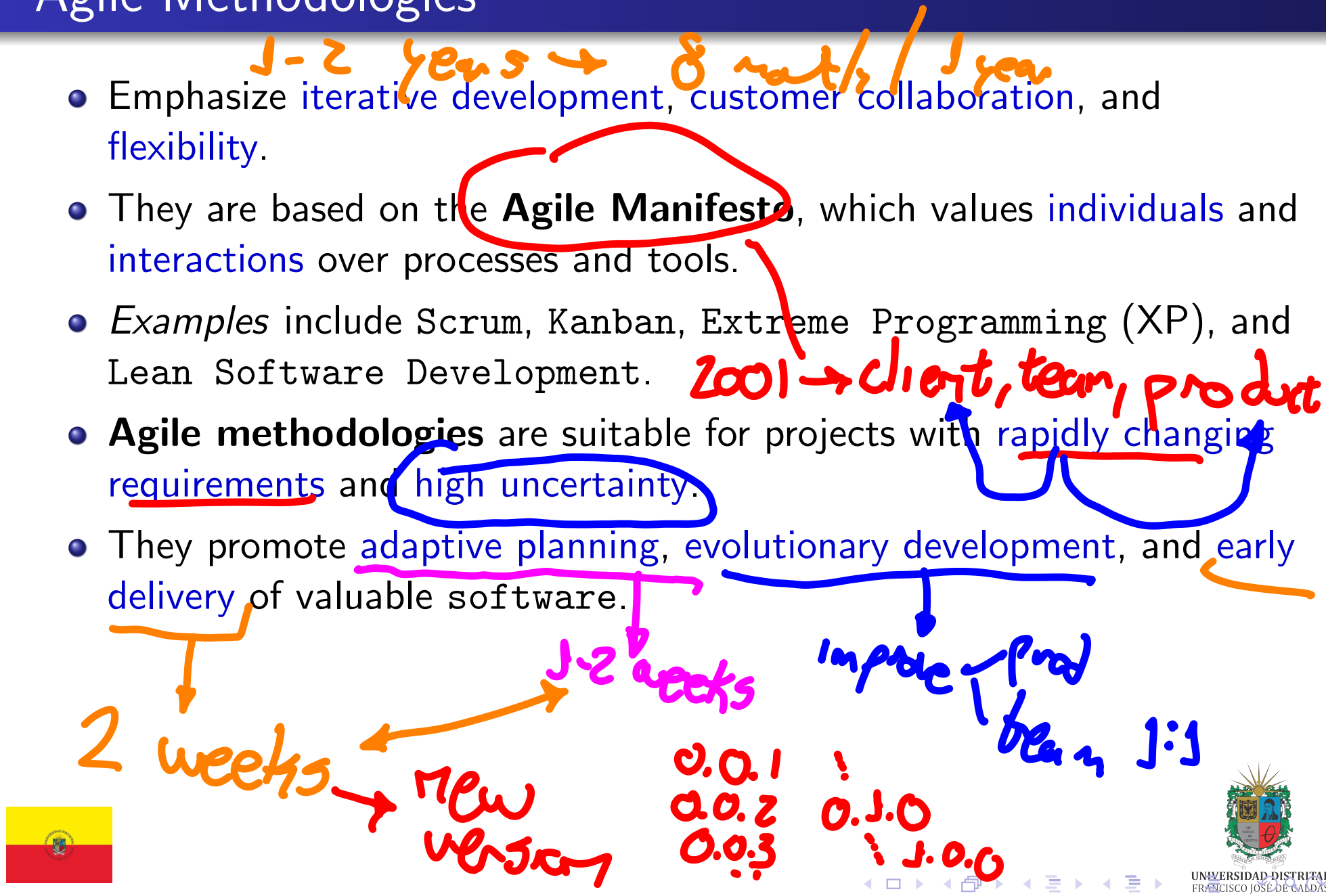
- Emphasize iterative development, customer collaboration, and flexibility.
- They are based on the **Agile Manifesto**, which values individuals and interactions over processes and tools.
- *Examples* include `Scrum`, `Kanban`, `Extreme Programming` (XP), and `Lean Software Development`.



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

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

Information Systems

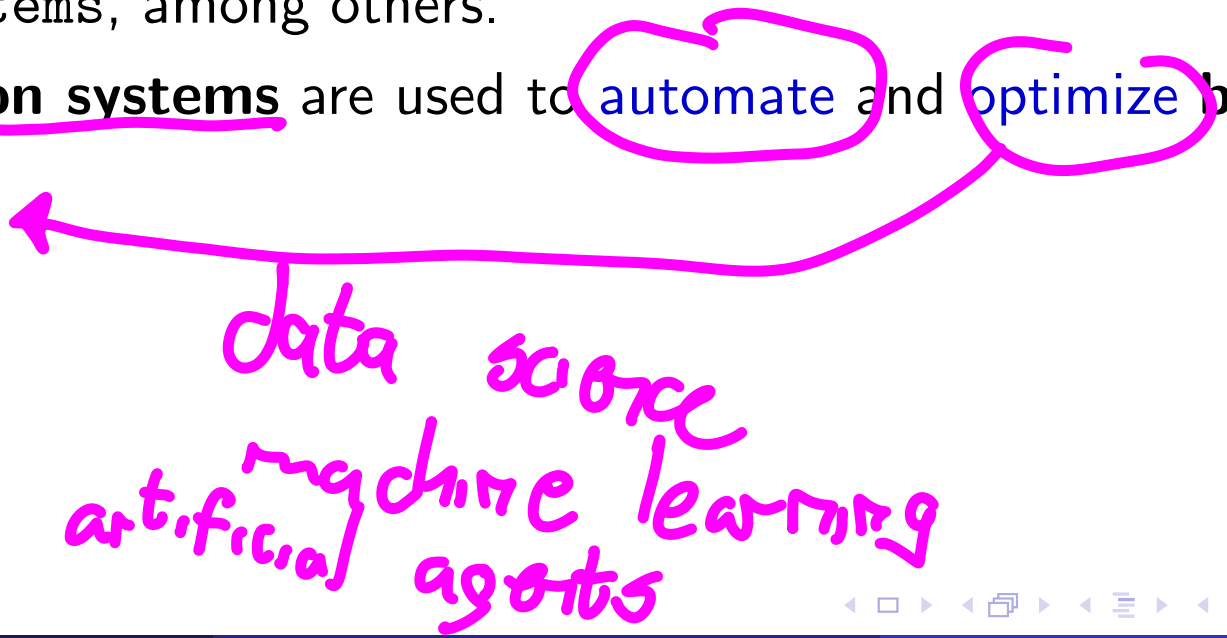
- An **Information System** is a system that *collects*, *processes*, stores, and disseminates **information**.
- **Information systems** are used to **support** and **manage** business operations.
- Examples of information systems include transaction processing systems, management information systems, 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**.

Information Systems

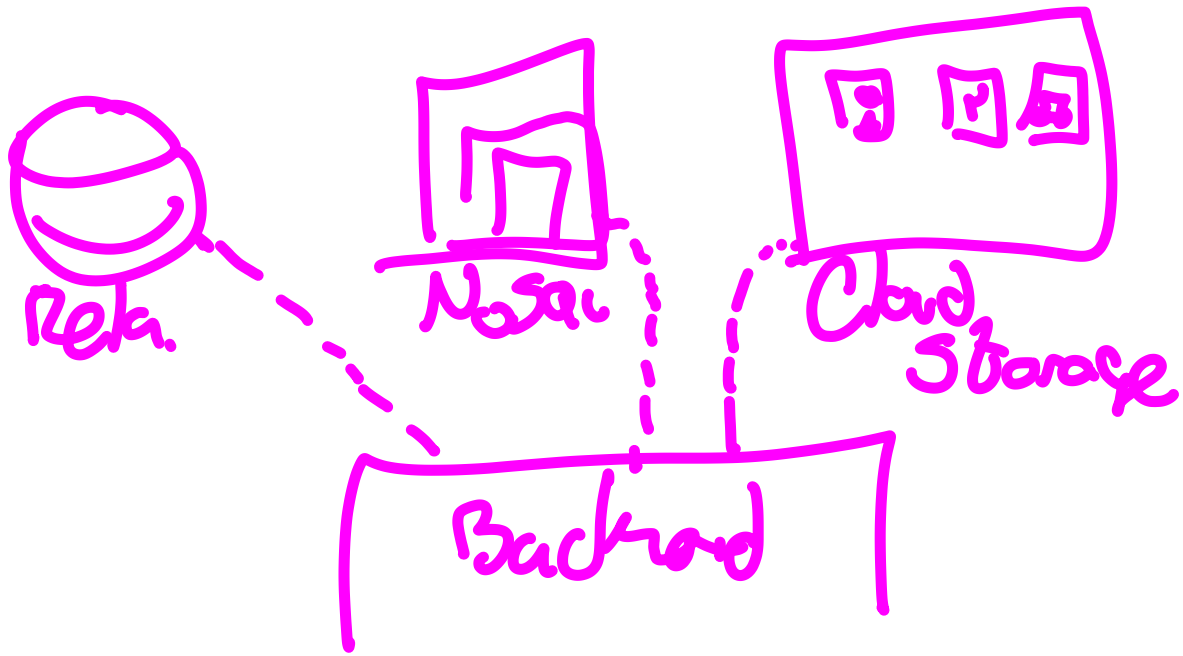
- An **Information System** is a system that *collects*, *processes*, stores, and disseminates **information**.
- **Information systems** are used to **support** and **manage** business operations.
- Examples of information systems include `transaction processing systems`, `management information systems`,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



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

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.

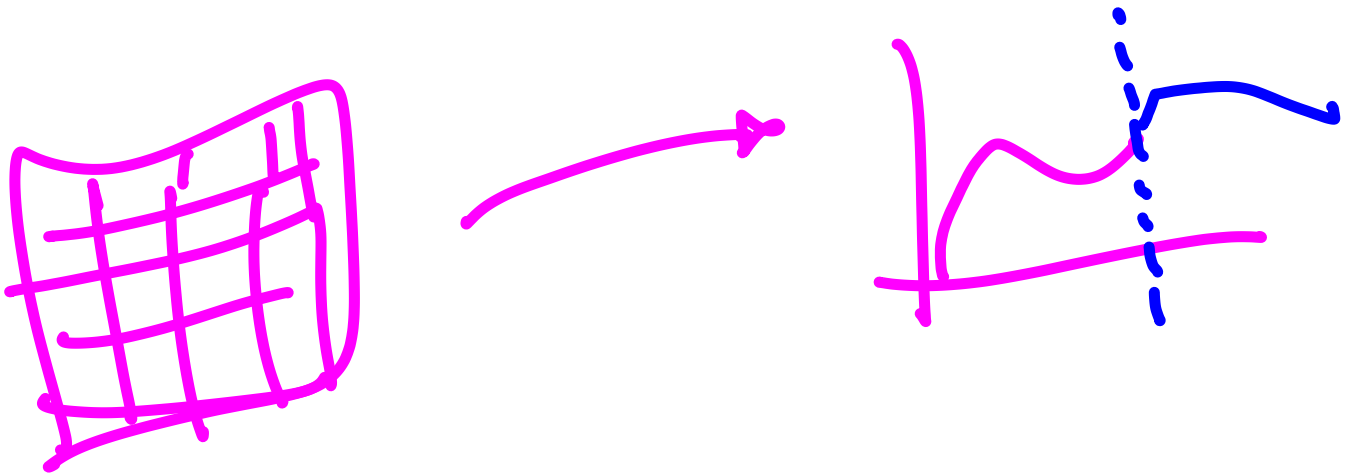


Expert Systems

- 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.

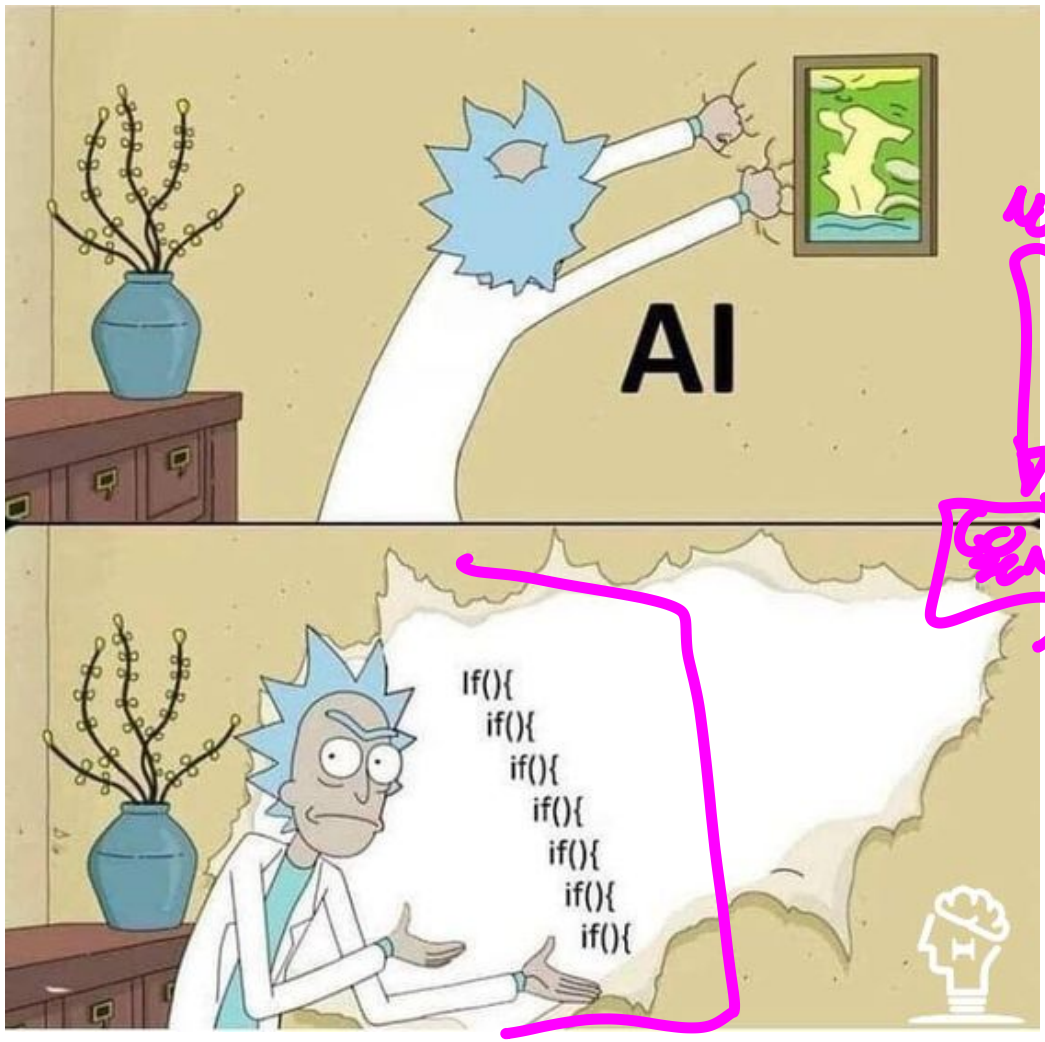
Expert Systems

- 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**.



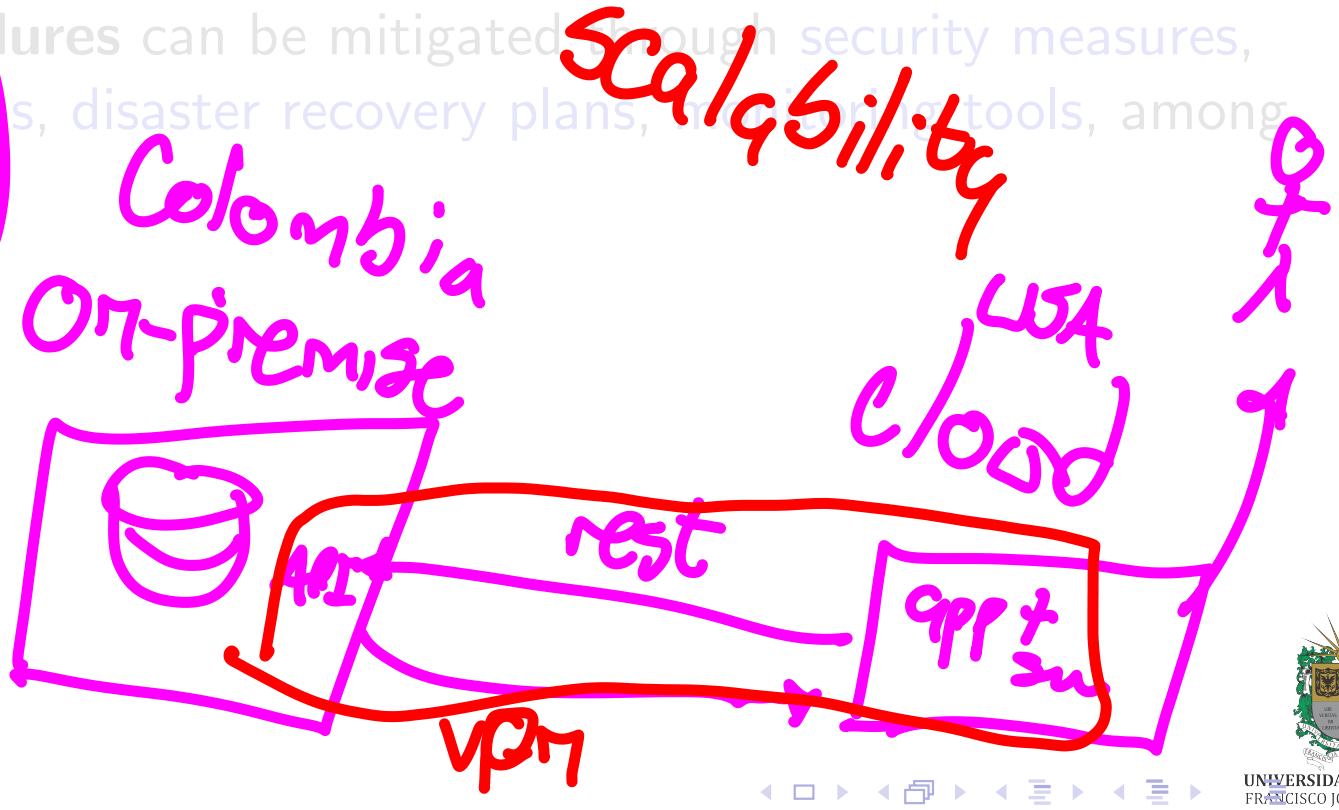
Expert Systems as Classical Artificial Intelligence

Here there is a great example of a diagnostic system.



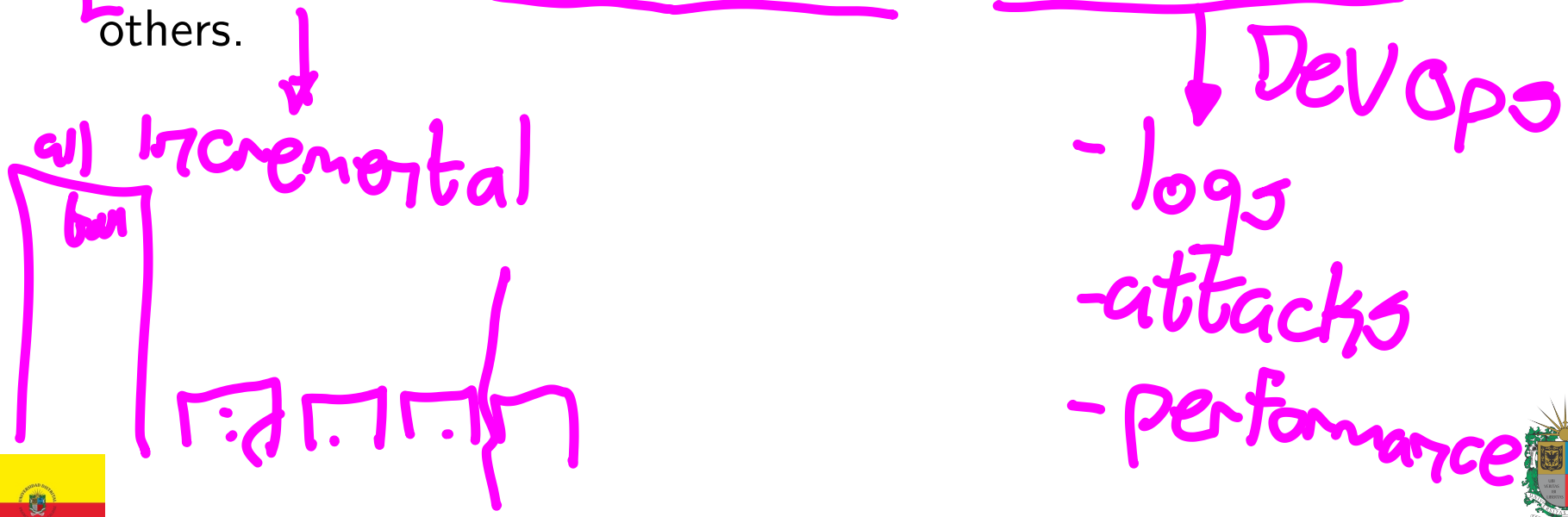
Risks and Failures in Information

- **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.



Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Thanks!

Questions?

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

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS