Introduction to Object-Oriented Modelling 🖥️

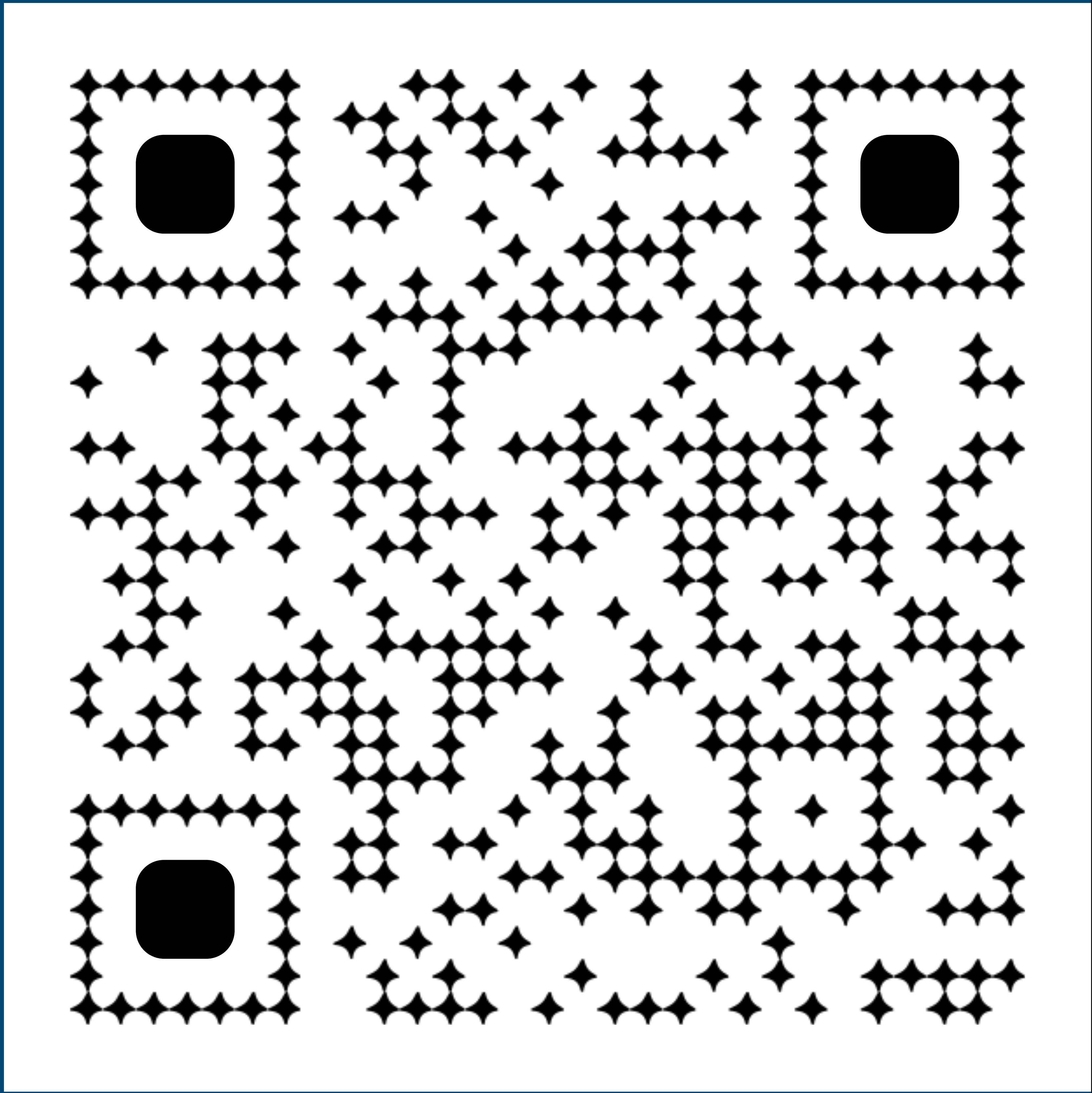
👨‍🏫 Dr. Ashish Sai

📖 BCS1430

🏛️ EPD150 MSM Conference Hall

📅🎓 Week 1 Lecture 1

Join the Discord Server



Welcome to BCS1430!

About Us

(Recap)

Dr. Ashish Sai

Assistant Professor

Department of Advanced Computing Sciences

📍PHS1 C4.005

📧ashish.sai@maastrichtuniversity.nl

💻ashish.nl

Past employment

- Expert Group Member - Crypto Sustainability, **World Economic Forum**
- Research Scholar - **University of California, Berkeley**
- Lecturer - **University of Amsterdam**
- Teaching Fellow - **Trinity College Dublin**

WORLD ECONOMIC FORUM

Berkeley UNIVERSITY OF CALIFORNIA

UNIVERSITY OF AMSTERDAM

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Dr. Tony Garnock-Jones

Assistant Professor

Department of Advanced Computing Sciences

📍PHS1 C4.032

📧tony.garnock-jones@maastrichtuniversity.nl

💻leastfixedpoint.com

- Tony works on bringing concepts from computer networking into programming languages and vice versa. He holds a Ph.D. from Northeastern University (Massachusetts, USA).

Teaching Assistants

- **Head Tutor: Osta, Nick van**
  - Merhi, Wissam
  - Ros Rodrigo, José
  - Panghe, Ada
  - Sella, Adee
  - Skog, Mikael
  - Poliakov, Ivan
  - Marcon, Daniel

Feedback from Introduction to Computer Science

Feedback Action Items for me

- Talk a bit slower 🚧
  - Please help me with this, interrupt me when I speak fast.
- New Hair Cut 🙄
- Do not ask us to ask TA for help 😓
  - Not possible for me to answer all the questions in the lab on my own (the reason we have TAs)
- Lecture slides with less pages ✅ [1]

[1] If you ever have issues with my slides (odd formatting, missing bits, not print friendly), please let me know, I can regenerate them to suit your requirements very easily. They are all just `Markdown` documents!

Feedback from the last run of OOM

Feedback Action Items for me

- More useful labs ✅
- (More) Transparent Grading Scheme for Project ✅
  - Project specification now contains a detailed grading rubric.
- Better Jokes 🤷🏻‍♂️
  - I wish!
- Clearly tell us what is important for exam ✅
  - Everything covered in the lectures, tutorials and labs (Sorry 😔)
  - I will give you a list of intended learning objectives before each lecture.
- 5 Minutes break is too short ✅
  - Fine, you can have longer breaks (6 minutes!)

Intended Learning Outcomes (Lecture 1)

1. Recognize how simple computational rules can lead to complex software behaviors.
2. Understand the key phases of the software development lifecycle and their importance.
3. Appreciate the role of design principles in handling evolving requirements.
4. Identify fundamental concepts of object-oriented modeling (e.g., UML, OOP basics).

Introduction

Part 2/5

The Paradox of Simple Rules and Complex Outcomes

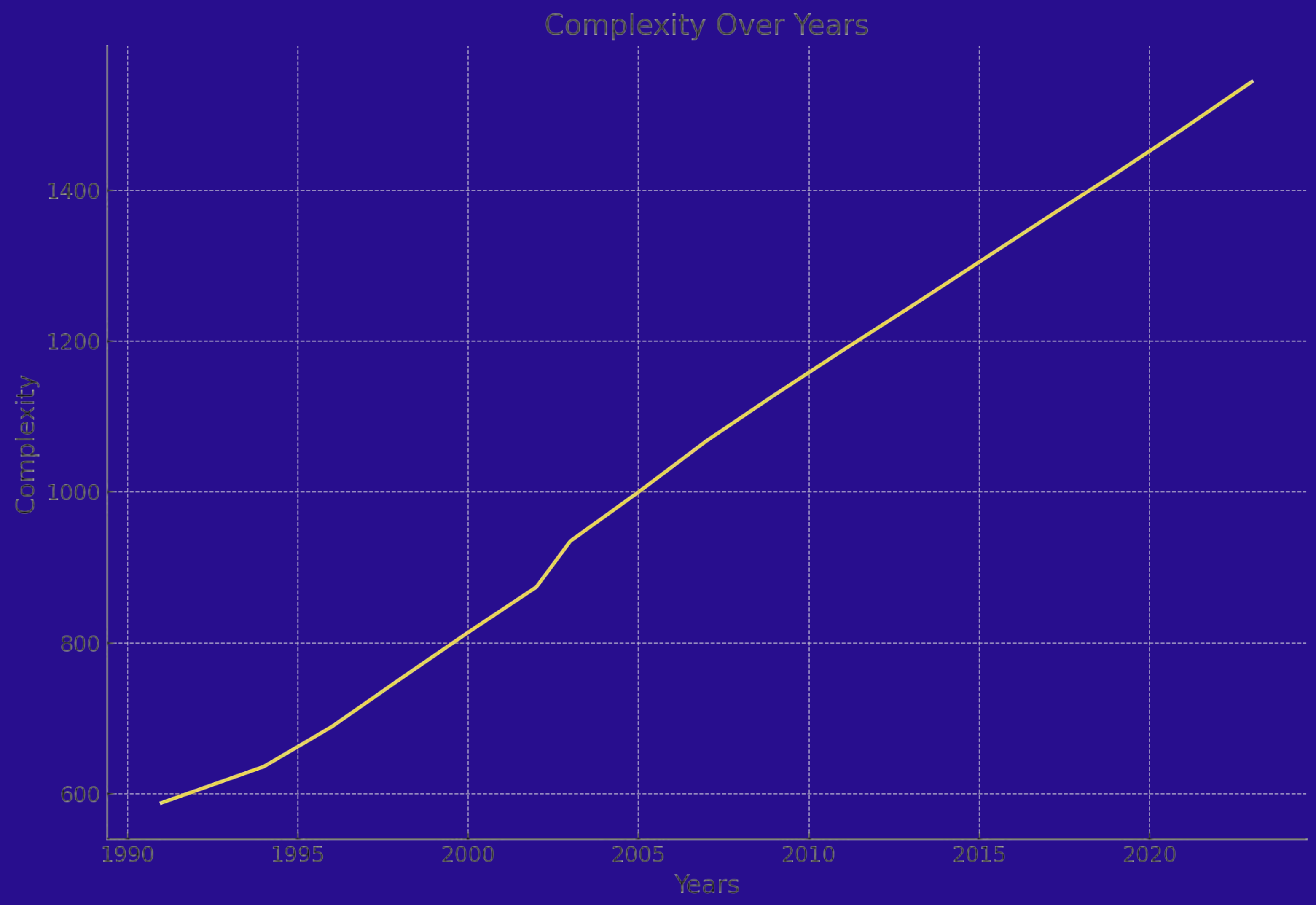
Simple Rules and Complex Outcomes

- Simple computational rules lead to **complex software behavior**.
  - Like Dutch 🇳🇱 traffic lights 🚦: sensing loops detect 🚲 bikes or 🚗 cars. No bike? Cars get green. Bike approaching? Cars stop, bike crosses—simple rules, yet complex traffic flow!

Simple Rules and Complex Outcomes

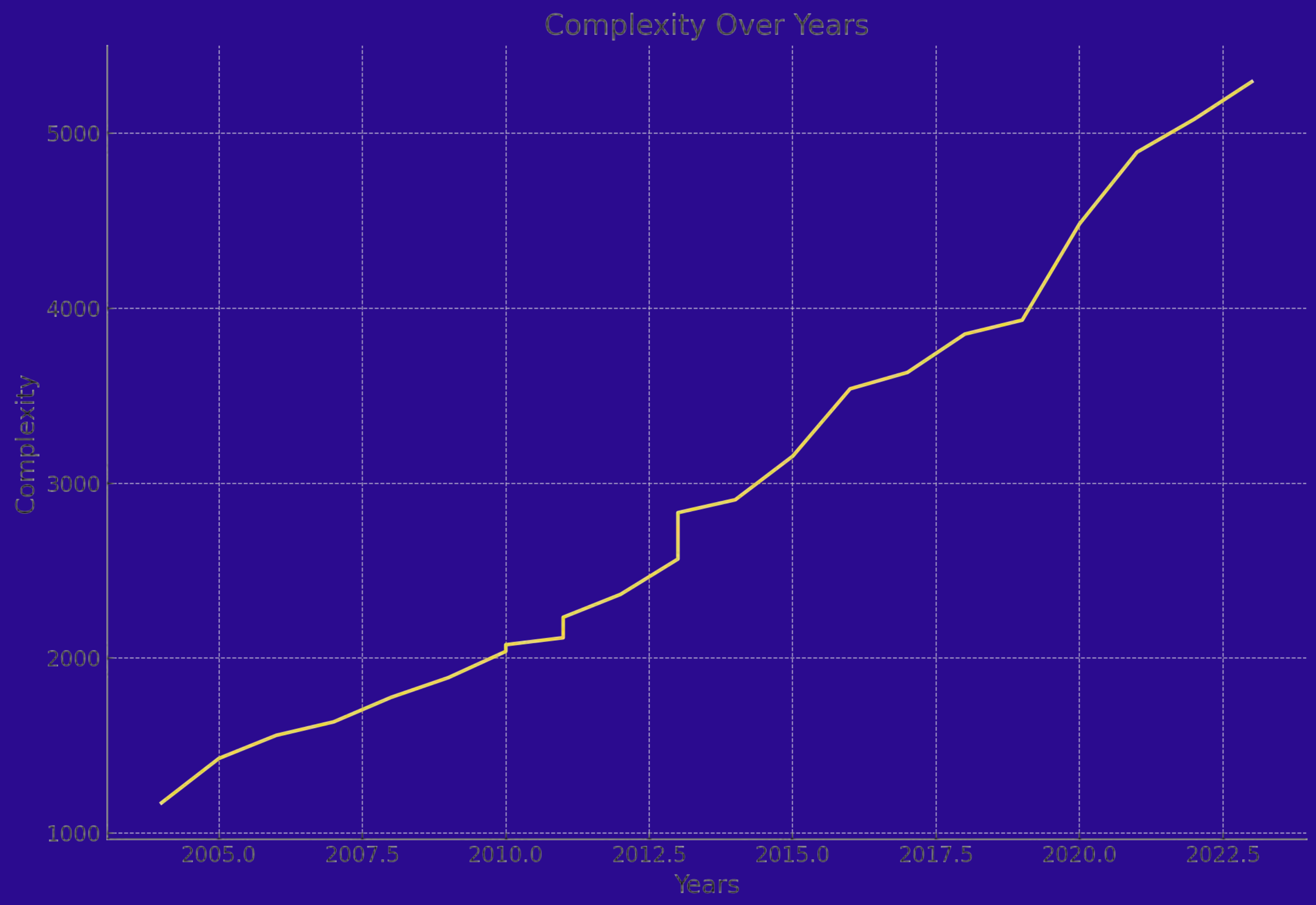
- This complexity is due to the interactions between simple steps, which become unpredictable as the software scales.
- **Human limitations**: We can create larger programs than we cannot fully *comprehend*, especially in *large*, *changing* teams.

Linux Kernel Complexity Over Time [1]



[1] (Source code: https://github.com/torvalds/linux)

Firefox Browser Complexity Over Time [1]



[1] (Source code: https://searchfox.org/mozilla-central/source)

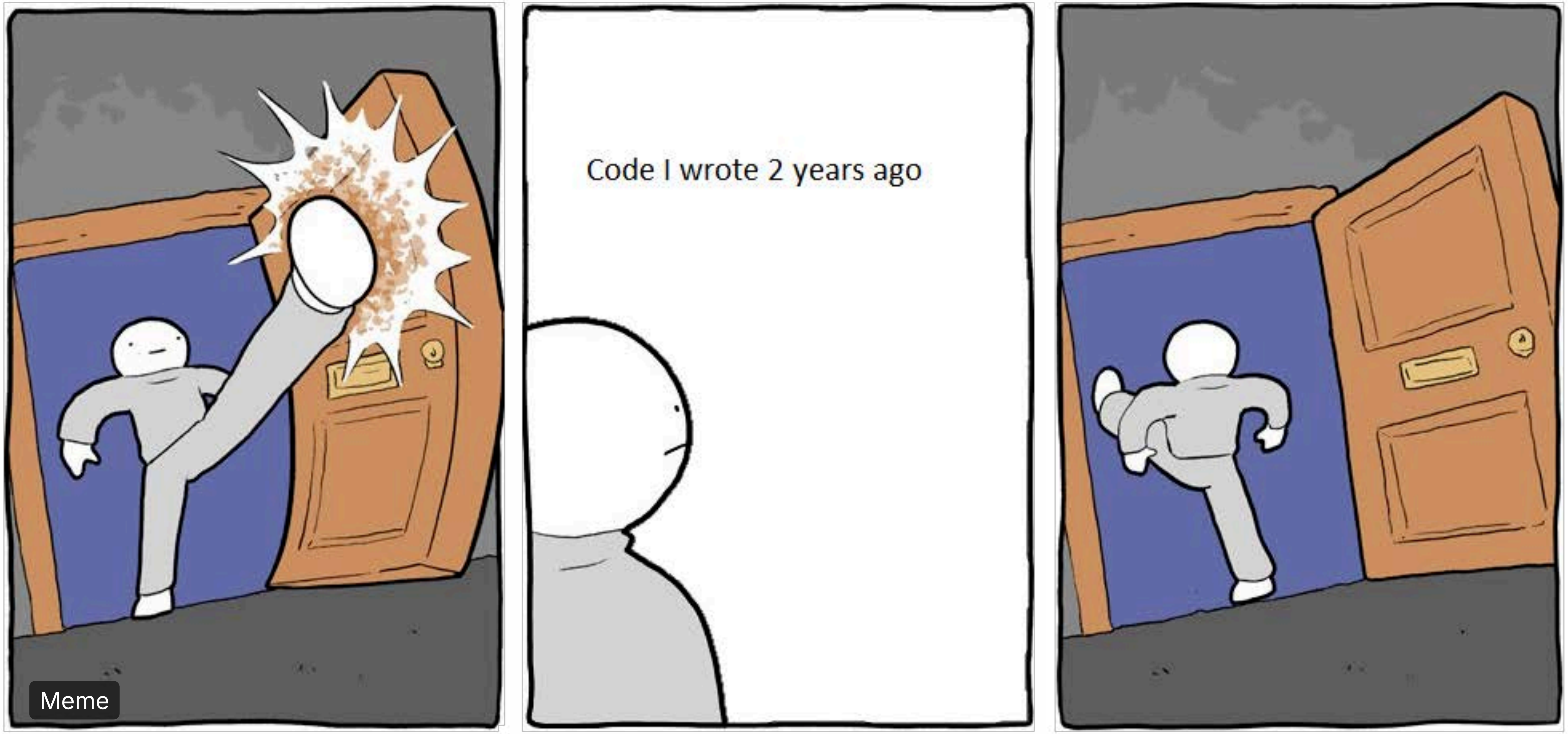
You create software for a purpose (*aka* requirements)

The Challenge of Evolving Software Requirements

- Software development is *dynamic*, with **evolving needs.**
  - **Example**: An e-commerce app starts with product listings but later adds AI recommendations or AR previews to meet market demands.

The Challenge of Evolving Software Requirements

- This evolution necessitates *flexible*, *adaptable designs* that can accommodate such changes without **major overhauls**.


Code I wrote 2 years ago
Meme

Designing for Complexity and Change

- Design principles help manage **complexity** and **changing needs**.
  - Object-oriented design uses techniques like **information hiding**, **interfaces**, and **polymorphism** to ensure ***loose coupling*** [1] , making components easier to reuse or replace.

[1] you will learn about loose coupling next week.

Good Software - Beyond Correctness

- Good software is more than just being **correct**!
  - It balances **efficiency** and **maintainability**.
  - **Example**: A fast algorithm may be **hard to modify**, while a simpler one might work better for **collaboration.**

Design as an Art and Science

- Software design is as much *an art as it is a science.*
- **Think**: *Design a music streaming application for older adults.*

Hip Hop Nana

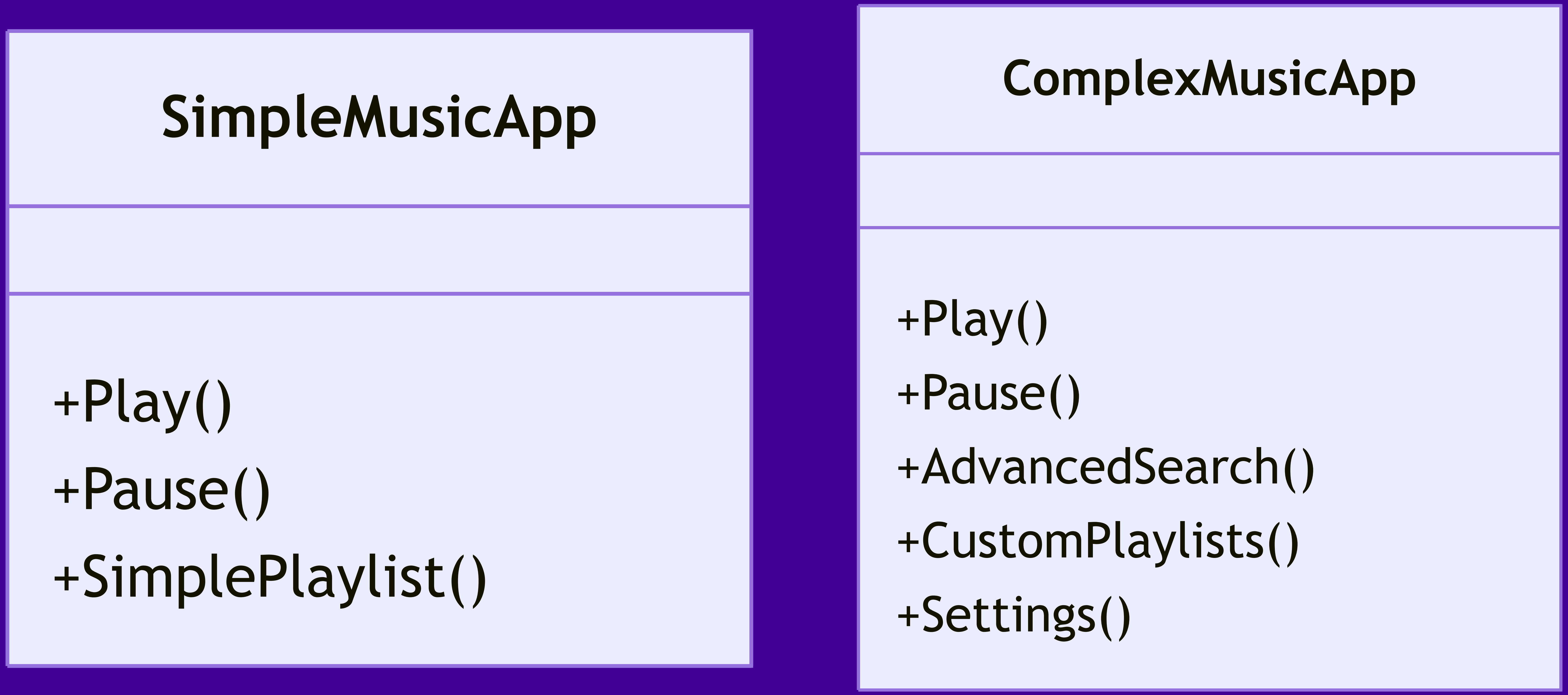
Option 1: Simple

- **Ease of Use:** Easy navigation with large, clear buttons.
- **Essential Functions:** Play, pause, and simple playlist management.
- **User Comfort:** Familiarity and comfort for users not accustomed to technology.
- **Development:** Easier implementation.

Option 2: Feature-Rich

- **Comprehensive Features:** Advanced search, custom playlists, and varied settings.
- **User Engagement:** Options for more tech-savvy users.
- **Competitive Edge:** A wider array of features.
- **Development Challenge:** More complex maintenance.

Hip Hop Nana (UML Examples)



SimpleMusicApp

+Play()
+Pause()
+SimplePlaylist()

ComplexMusicApp

+Play()
+Pause()
+AdvancedSearch()
+CustomPlaylists()
+Settings()

How to design good software?

Ride Sharing Apps

- Apps like ***Uber***, ***Bolt***, and ***Lyft*** need to adapt quickly to new technologies (~~LLM, LLM, LLM~~) and user expectations.

Uber's Platform

- **Initial Approach:**
  Began as a service for booking luxury car rides in **San Francisco** 🌉.

- **Evolution:**
  Expanded operations to over **900** metropolitan areas 🌍.

- **Diversification:**
  Added services like *UberX*, *UberPOOL*, and *UberEats*.

Market and Technological Changes

- **User Feedback:**
  Introduced *in-app tipping*, leading to billions 🤑 in tips.

- **Regulatory Adaptation:**
  Adjusted operations to comply with local laws.

- **Technological Advancements:**
  Integrated AI/machine learning to optimize millions of trips daily.

So how do Uber and other companies design good software?

> With the principles and practices of **Software Engineering**, a discipline dedicated to crafting high-quality software!

programmer

software engineer

(Very Brief) Introduction to Software Engineering

Software Engineering

- **Definition**: Application of **engineering principles** to **software development**.
- **Goal**: Produce `reliable`, `efficient`, `maintainable`, and `usable` software.

The Essence of Software Engineering

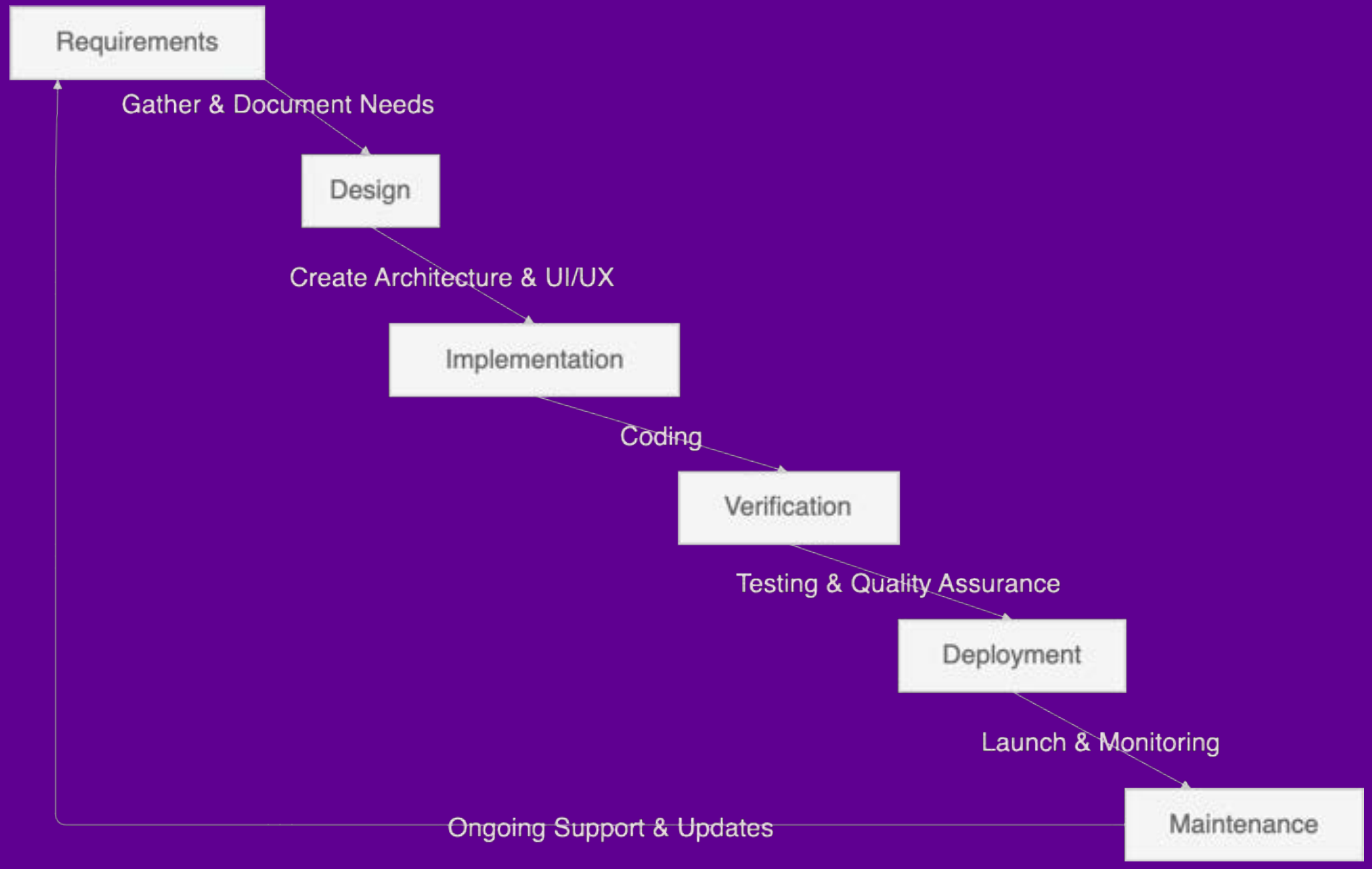
- **Not** just about coding. It's about:
  - Understanding user needs (***Requirements Engineering***)
  - Designing `robust`, `scalable`, `secure` systems
  - Managing the development process

> It integrates aspects of **computer science**, **project management**, and **engineering**.

Software Development Lifecycle

- A process used to develop software
  - **Waterfall Model:** A sequential model with phases
  - **Requirements → Design → Implementation → Verification → Deployment → Maintenance**

Waterfall Model Overview



Requirements

Gather & Document Needs

Design

Create Architecture & UI/UX

Implementation

Coding

Verification

Testing & Quality Assurance

Deployment

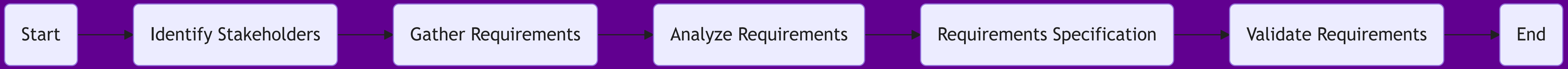
Launch & Monitoring

Maintenance

Ongoing Support & Updates

Requirements Phase

- Provides a clear foundation for the project.
- **Example**: Gathering requirements for a grocery app.



- Two types of requirements: ==**Functional** ==and **Non-Functional**.

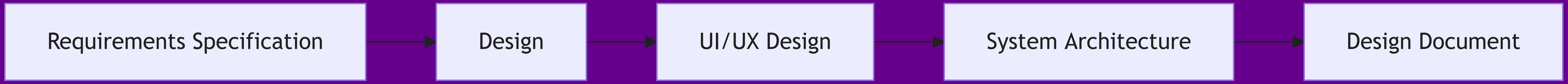
Requirements Phase: Functional & Non-Functional Requirements[1]

- **Functional Requirements**: Define what the software **must do** (user authentication, payment processing, etc.).
- **Non-Functional Requirements**: Specify performance, security, and other "-ility" concerns.

[1] More on this next week!

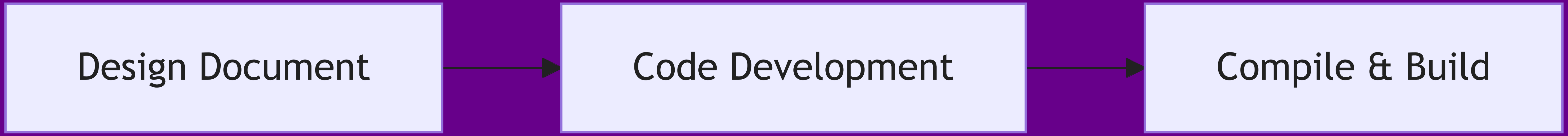
Design Phase

- Outlines how the app will **look and function**.
- **Example:** Detailed designs for the interface and system architecture.



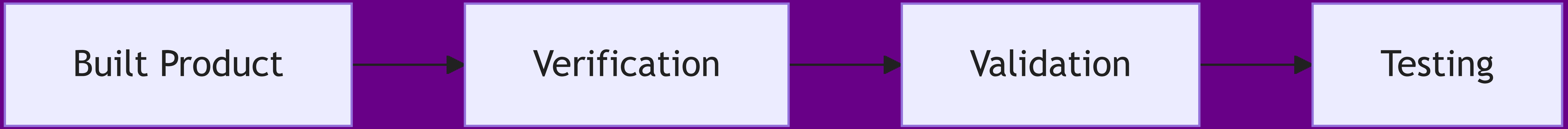
Implementation Phase

- Translates design documents into **working code**.
- **Example**: Writing and compiling the code.



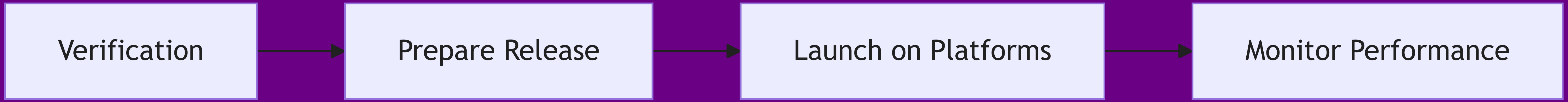
Verification Phase

- Ensures the app is built *correctly* and ready for deployment.
  - **Verification**: “Are we building the product right?”
  - **Validation**: “Are we building the right product?”
  - **Testing**: Running a program with selected data to uncover bugs.



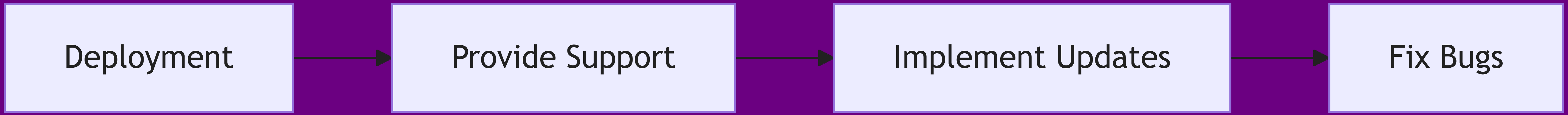
Deployment Phase

- The phase where software is released to end-users.
  - **Example:** Publishing the app on platforms and monitoring performance.



Maintenance Phase

- Keeps the app ***functional and up-to-date***.
  - **Example:** Ongoing support, bug fixes, and updates.

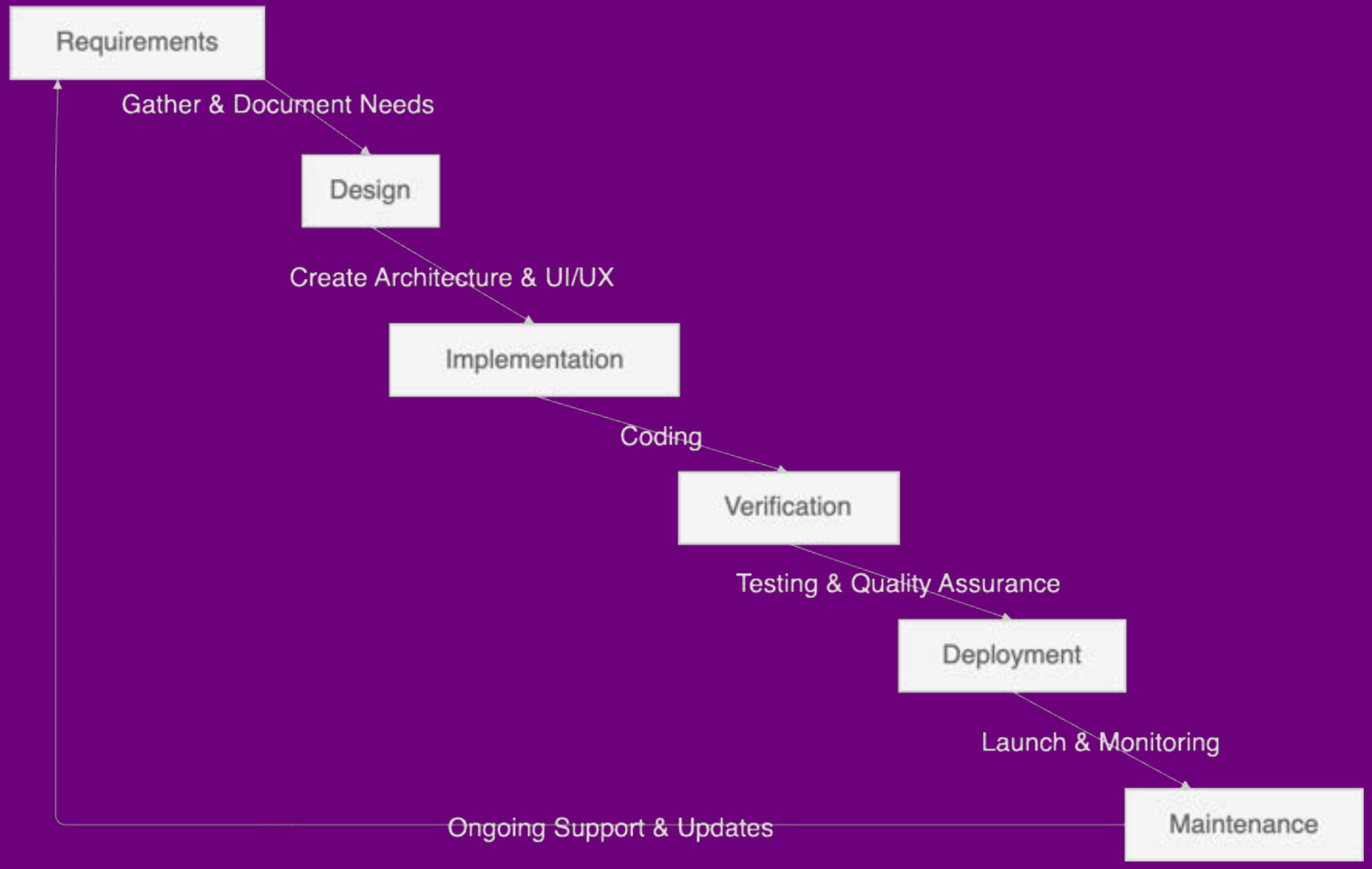


Deployment → Provide Support → Implement Updates → Fix Bugs

Short Break

Do not leave your seat (~~5 min~~ 6 min)

Waterfall Model Overview (Again)



Requirements

Gather & Document Needs

Design

Create Architecture & UI/UX

Implementation

Coding

Verification

Testing & Quality Assurance

Deployment

Launch & Monitoring

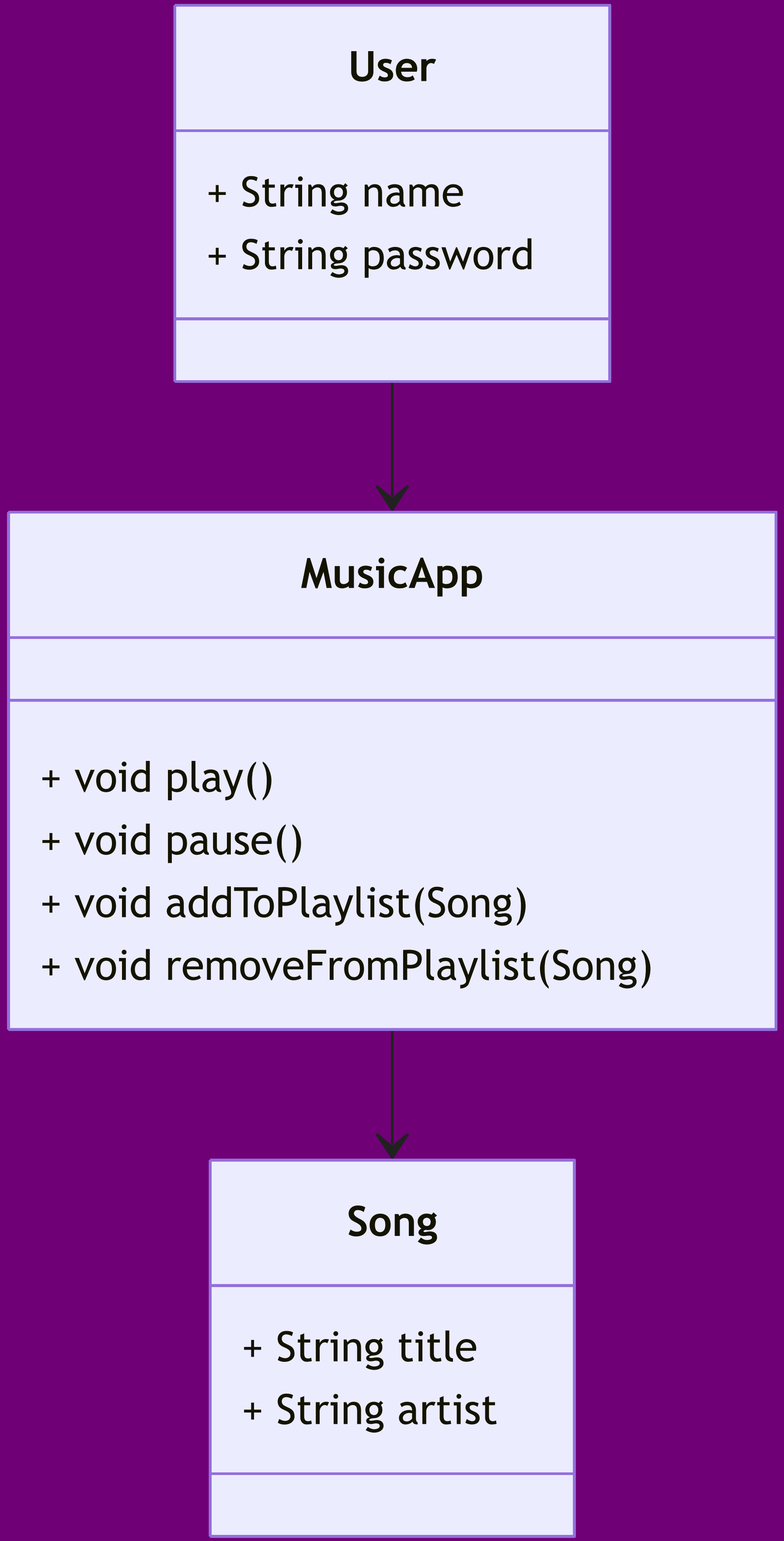
Maintenance

Ongoing Support & Updates

Do not forget the Granny's! Design their music app.

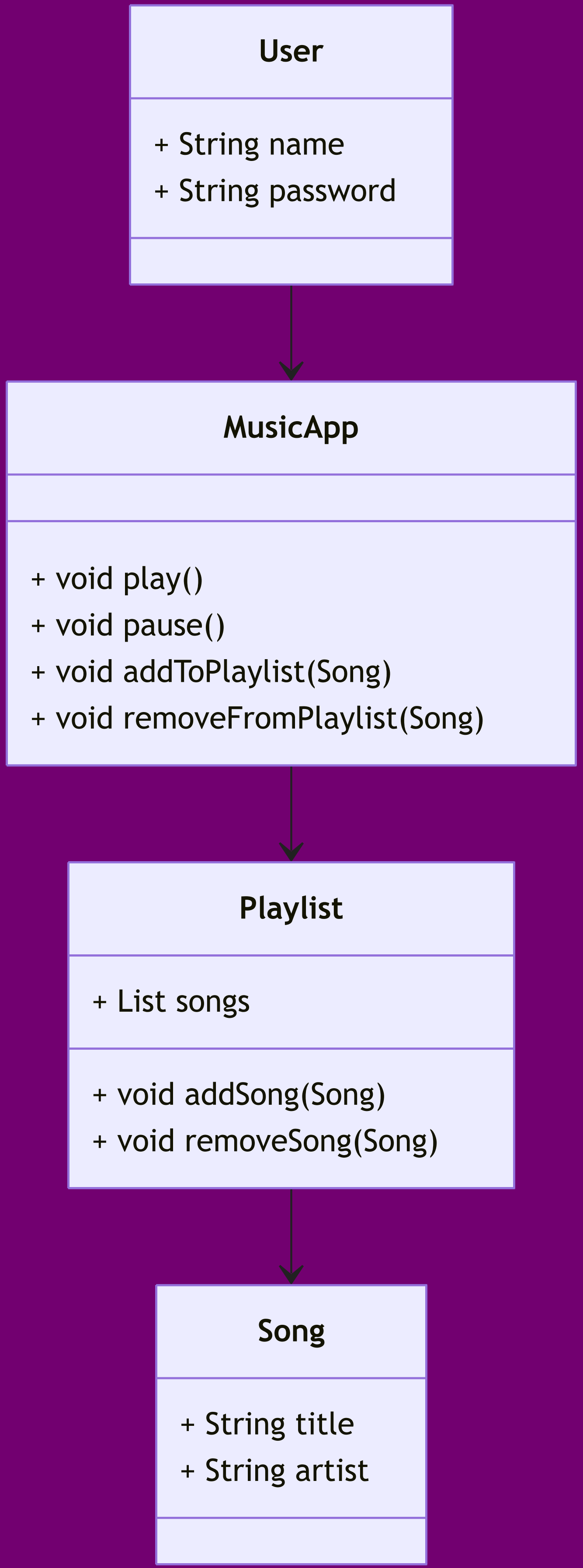
Requirements Phase for Hip Hop Nana

- **Activities:**
  - User interviews, surveys
  - Analyze feedback, document constraints
- **Deliverables:**
  - Requirements Specification Document



Key Requirements:

Type	Requirements
Functional	User login, music playback, playlist management
Non-Functional	Usability, performance, accessibility



User

+ String name
+ String password

MusicApp

+ void play()
+ void pause()
+ void addToPlaylist(Song)
+ void removeFromPlaylist(Song)

Playlist

+ List songs

+ void addSong(Song)
+ void removeSong(Song)

Song

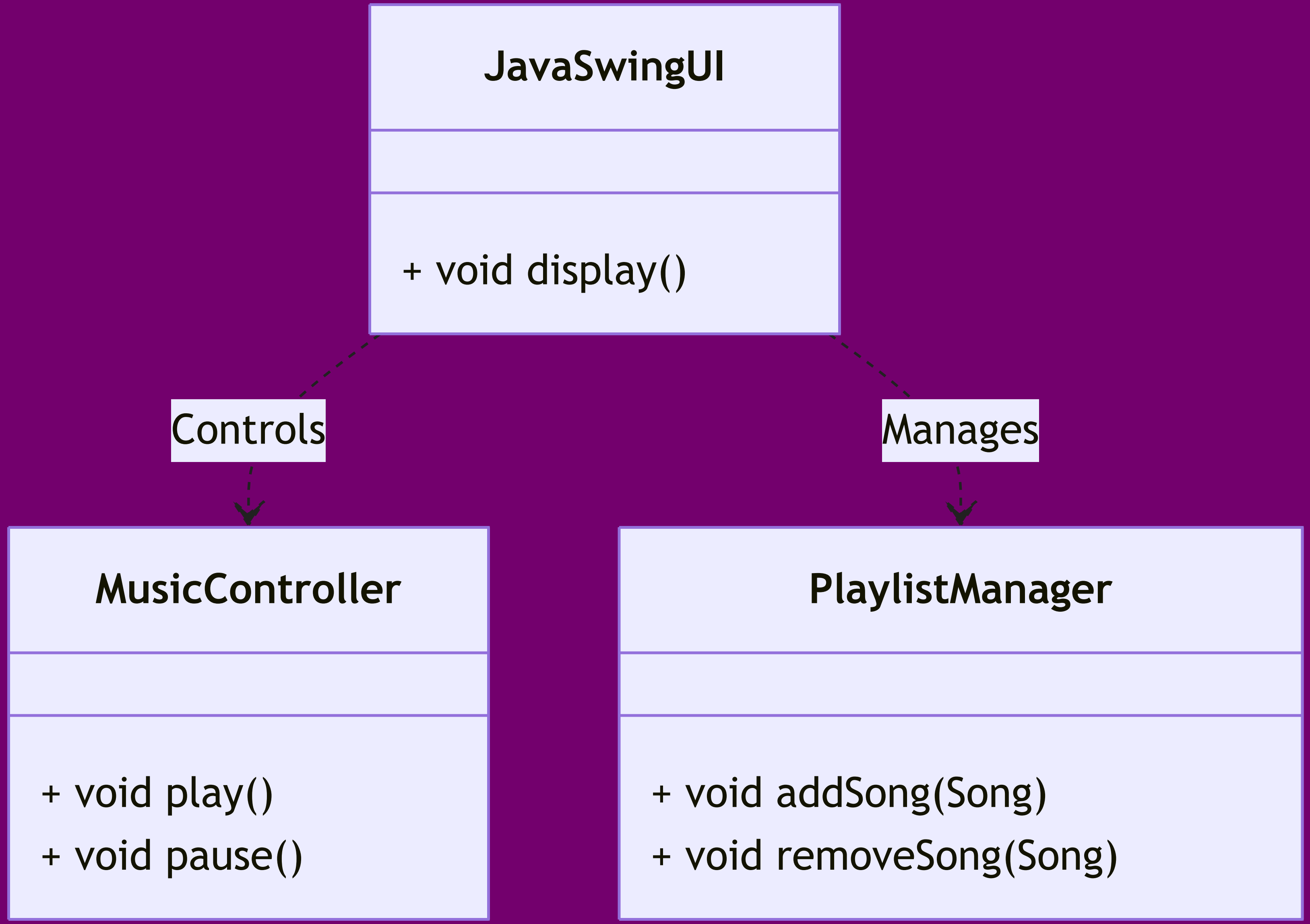
+ String title
+ String artist

Design Phase for Hip Hop Nana

- **Activities**:
  - UI/UX mockups for seniors
  - System architecture & database schema
- **Deliverables**:
  - Design Mockups, UML Diagrams

Implementation Phase for Hip Hop Nana

- **Activities:**
  - Write Java code based on designs
  - Core functionality tests
- **Deliverables:**
  - Source Code



Verification Phase for Hip Hop Nana

- **Activities**:
  - Test cases for usability, functionality
  - Validate performance on Java environments
- **Deliverables**:
  - Test Report

Test Type	Description
Usability	Ease of use for seniors
Functionality	Music control and playlist management
Performance	Response times, resource usage

Deployment Phase for Hip Hop Nana

- **Activities:**
  - Package Java app for distribution
  - Release and monitor feedback
- **Deliverables:**
  - Installation Package, Deployment Guide

Maintenance Phase for Hip Hop Nana

- **Activities:**
  - Bug fixes, user feedback
  - Regular updates
- **Deliverables:**
  - Updated App Versions, Maintenance Logs

Two more things

Importance of Good Design

- **Good design = successful software development**
  - Easier `maintenance` and `scalability`
  - Better `performance` and `UX`
- Poor design can lead to:
  - Increased costs and complexity
  - Hard-to-adapt code

The Role of Software Engineers

- **Software Engineers are like *architects***:
  - Analyzing user **requirements**
  - **Designing** and **implementing** solutions
  - Ensuring **quality** and **performance**

Quick Recap: Object-Oriented Programming (OOP)

In the next lecture, **Nick** will provide an OOP refresher.

Object-Oriented Programming (OOP)

- **What is OOP?**
  - A style of programming focused on “objects” **(data + behaviors)**.

Understanding Objects in OOP

- **Core Concept**: An object represents an entity with **identity, behavior, and state.**
  - **Example**: A `Book` object in a library system, with `title`, `author`, and a `checkAvailability()` method.

Classes — The Blueprint

- **Definition:** Classes define the blueprint for objects.
  - **Example:** A `Book` class with properties (`title`, `author`) and methods (`borrow()`, `return()`).

Introduction to Object-Oriented Analysis and Design (OOA/OOD)

OOA vs. OOD

- **Object-Oriented Analysis (OOA)**: Identifies objects and their interactions in the *problem* or *system*.
- **Object-Oriented Design (OOD)**: Proposes a conceptual solution (blueprint) to meet requirements.

Hello, Unified Modeling Language (UML)

Unified Modeling Language (UML)

- *A graphical language* for **specifying**, **visualizing**, **constructing**, and **documenting** software.
- Standardized approach to design; helps communication.

Advantages of Using UML

- **Common language** for developers, clients, managers.
- Reduces **ambiguity**, aids in documentation.

Great, now I know diagrams + OOP. Am I coding wizard now?

Not yet!

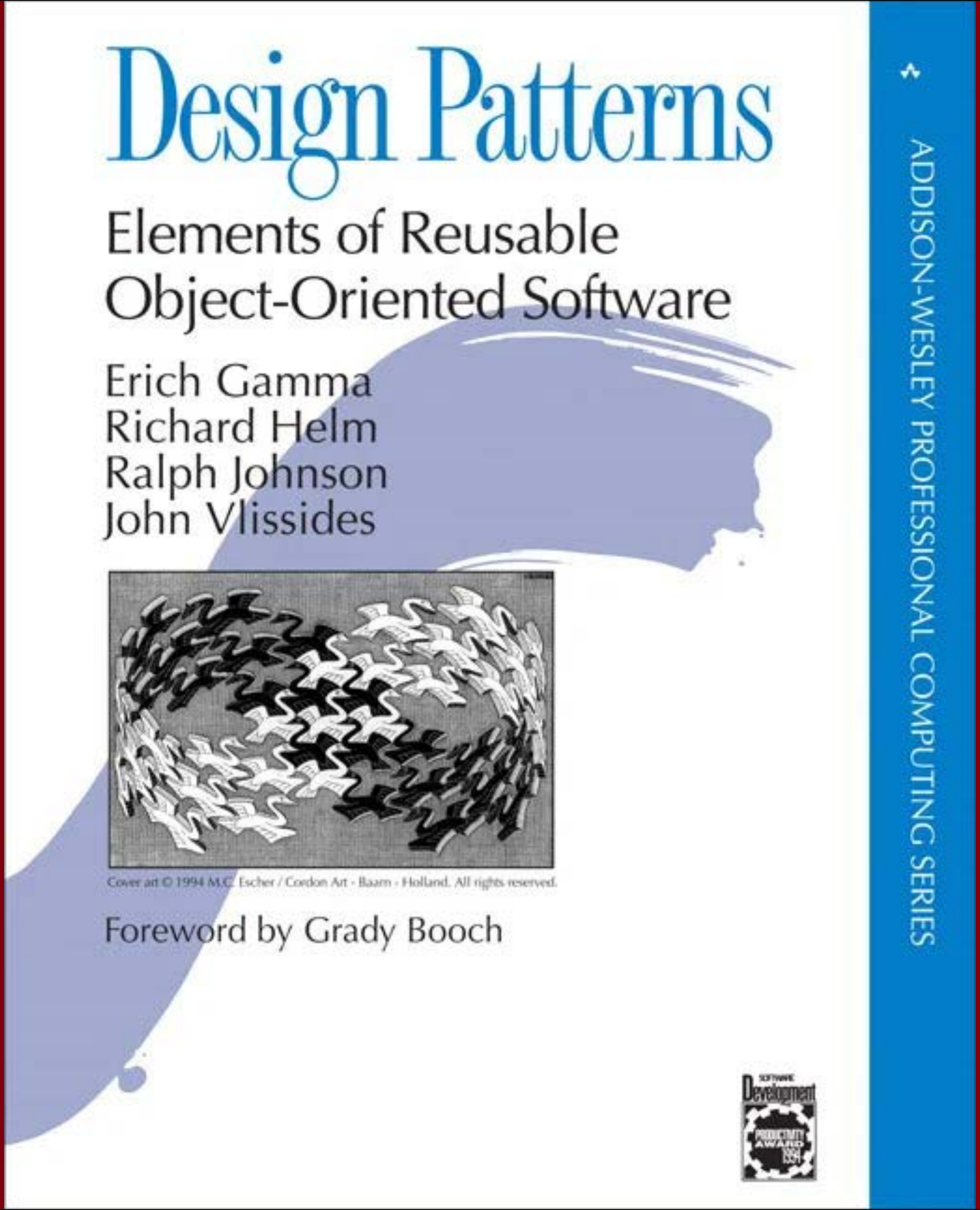
Introduction to Design Patterns

Design Patterns

- **Reusable solutions** to **common design problems**
- Provide **proven** techniques for architecture and design

Characteristics of Design Patterns

- Defines a **shared design vocabulary**
- **Language-agnostic**, widely applicable
- Encourages best practices (`modularity`, `separation of concerns`)



Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

History of Design Patterns

- **"Gang of Four"** (GoF) popularized them in *Design Patterns: Elements of Reusable Object-Oriented Software.*

I can use design patterns, am I a coding wizard now?

Not yet!

Introduction to Code Smells

Code Smells

- Patterns in code signaling **potential flaws**
- Examples:
  - **Long Method**: too many lines
  - **Large Class**: too many responsibilities
  - **Duplicate Code**: repeated blocks

The Knight Capital Catastrophe

- **2012 meltdown**: $460 million **lost** in 45 minutes
- Triggered by a **flawed deployment** activating old code
- **Dormant/obsolete code** = a classic code smell

Introduction to Code Refactoring

What is Code Refactoring?

- Restructuring existing code **without changing** its **external behavior**
- Improves ***internal structure***, ***readability***, ***maintainability***

Principles of Code Refactoring

- Keep changes **small** and **incremental**
- **Preserve functionality** at each step
- **Test** continuously to avoid introducing bugs

Course Overview

What You Will Learn in OOM

1. Understanding OOD/OOM
2. Code Improvement (refactoring)
3. UML (modeling systems)
4. Design Patterns (high-quality software)

Course Philosophy

- Moving from *writing basic programs* to designing (somewhat) **complex software**
- We focus on **OOP + software engineering** practices
- The project will let you apply these in a real scenario

Essential Concepts

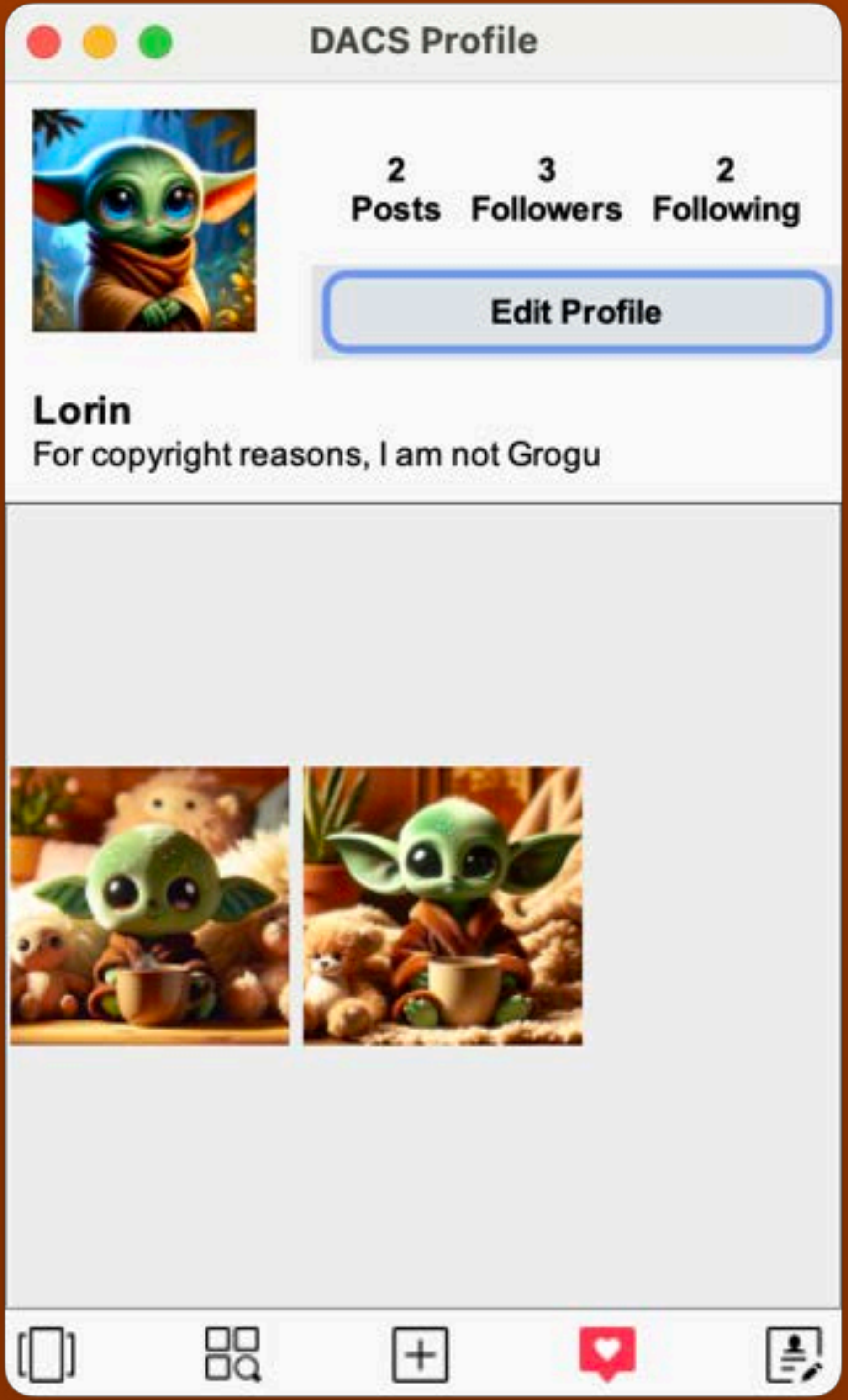
- Object-Oriented Design
- Code Smells & Refactoring
- Modelling with UML
- Design Patterns

Weekly Breakdown

Topic	Lectures	Lab
Week 1: Introduction	2 Lectures	1 Lab
Week 2: Object Oriented Design	2 Lectures	1 Lab
Week 3: Code Refactoring & UML	2 Lectures	1 Lab
Week 4: Design Patterns	2 Lectures	1 Lab
Week 5: Design Patterns	2 Lectures	1 Lab
Week 6: Exam Prep	2 Q&A	1 Lab

Course Project (🐥)

Quackstagram



Lorin
For copyright reasons, I am not Grogu

> Obviously not inspired by any similar-sounding app.

Grading

Assignment	Points	Percent
Quackstagram 🐥 Project	30	30%
Final Exam	70	70%
Total	100	—

You need >55% total to pass.

Grade	Range
10	> 95% - <= 100%
9	> 85% - < 95%
8	> 75% - < 85%
7	> 65% - < 75%
6	> 55% – < 65%
F	<55%

What do we expect from you?

Programming Expectations

- You have completed **BCS1120 (Procedural)** & **BCS1220 (Objects in Programming)**
- Comfortable with **Java** ☕ & OOP
- Expect to write (way) more code than in BCS1110 (Intro to CS)

Attendance and participation

- You are ***expected*** to attend Monday & Tuesday lectures + Friday labs

Course Material

- Additional readings on the course page (no purchase necessary)
- Java + VSCode (following BCS1120 setup)

Important pep talk!

- You **can** and ***will*** succeed in this class
- I will do my best to help you learn everything you hope to learn

Support

Support from me

- I will do what I can to help you complete the course successfully
- If you need help or more time, please let me know; no judgment

- **Do not suffer in silence.** Talk to me if you feel stuck or behind. I promise to work with you.

Student hours ⌛

- I have student hours (office hours) every Monday from 14–15 in PHS1 C4.005
- Come by with any questions you have

Course Policies

Simple: Be Kind, Be Nice and Be Considerate

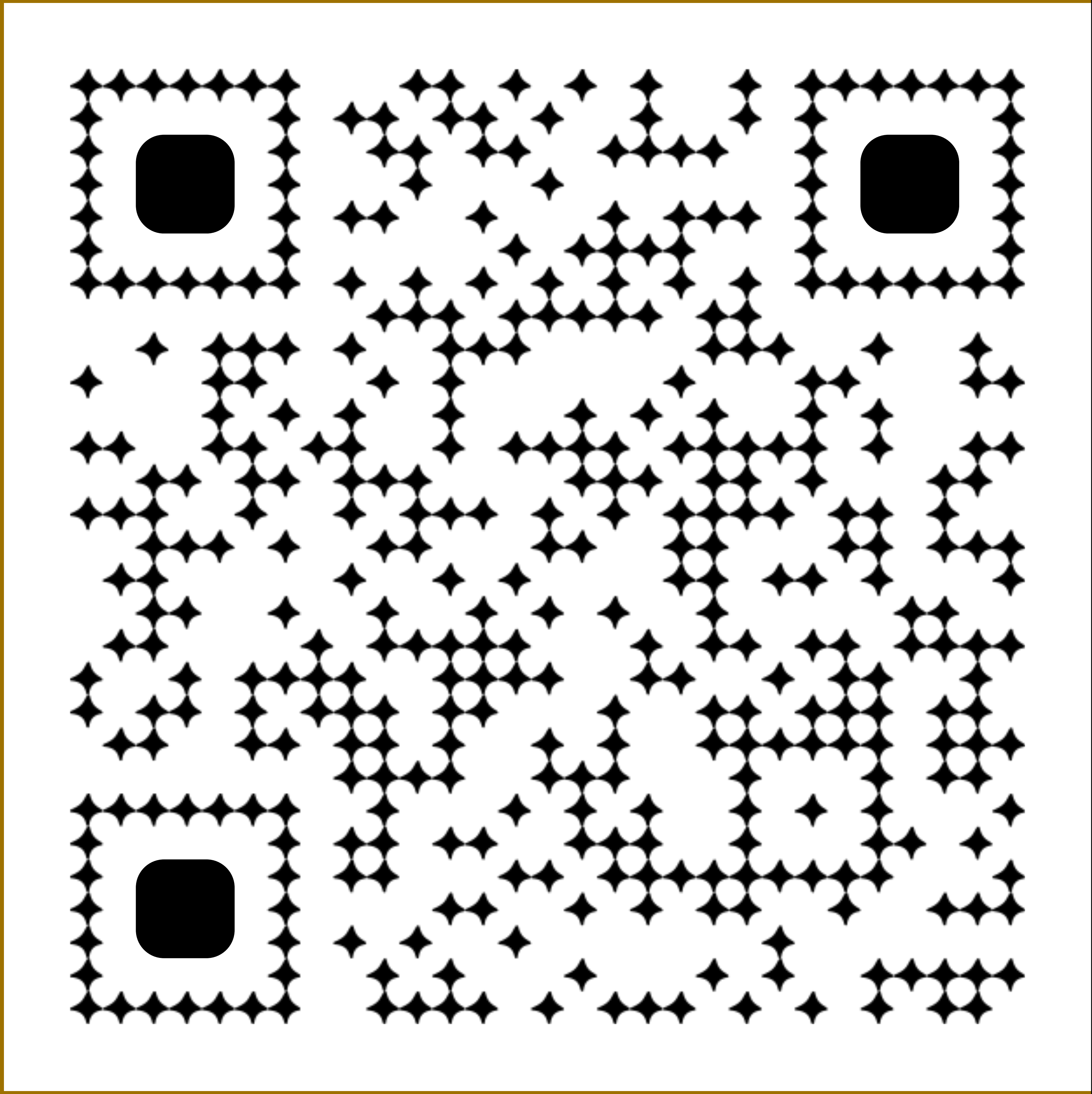
Class Policies

- No discrimination or violence tolerated
- Academic Honesty per UM Policy
- Special Needs? Please talk to me

Course Communication

- UM Canvas
- Discord Server
- Email

Join Discord Server (if you haven't yet)



Remember: I am here to support you if you need it.

👋 Let's have a great semester!