Welcome to the
CoGrammar
Software Design

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**

- No question is daft or silly - **ask them!**

- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.

- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

CoGrammar

Software Engineering Session Housekeeping cont.

- For all **non-academic questions**, please submit a query:

  www.hyperiondev.com/support

- We would love your **feedback** on lectures: **Feedback on Lectures**

CoGrammar

Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:

Ian Wyles
Designated Safeguarding Lead

Simone Botes

Rafiq Manan

Charlotte Witcher

Nurhaan Snyman

Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar   HyperionDev

Skills Bootcamp Progression Overview

To be eligible for a certificate of completion, students must fulfil three specific criteria. These criteria ensure a high standard of achievement and alignment with the requirements for the successful completion of a Skills Bootcamp.

✅ Criterion 1 - Meeting Initial Requirements

Criterion 1 involves specific achievements within the first two weeks of the program. To meet this criterion, students need to:

- Attend a minimum of 7-8 hours per week of guided learning (lectures, workshops, or mentor calls) within the initial two-week period, for a total minimum of 15 guided learning hours (GLH), by no later than 15 September 2024.

- Successfully complete the Initial Assessment by the end of the first 14 days, by no later than 15 September 2024.

CoGrammar

Skills Bootcamp
Progression Overview

✅ **Criterion 2 - Demonstrating Mid-Course Progress**

**Criterion 2 involves demonstrating meaningful progress through the successful completion of tasks within the first half of the bootcamp.
To meet this criterion, students should:**

- Complete 42 guided learning hours and the first half of the assigned tasks by the end of week 7, no later than 20 October 2024.

CoGrammar

Skills Bootcamp Progression Overview

✅ **Criterion 3 - Demonstrating Post-Course Progress**

Criterion 3 involves showcasing students' progress after completing the course. To meet this criterion, students should:

- Complete all mandatory tasks before the bootcamp's end date. This includes any necessary resubmissions, no later than 22 December 2024.

- Achieve at least 84 guided learning hours by the end of the bootcamp, 22 December 2024.

CoGrammar

Learning Objectives & Outcomes

- Analyse use cases to capture user requirements, identifying actors and scenarios.

- Design sequence and class diagrams to illustrate object interactions and their relationships within programs.

- Apply CRUD matrices to represent entity permissions and operations.

- Develop Python programs that implement the MVC pattern

CoGrammar

Building the Foundation: Essential Software Engineering Concepts

You've already learned essential concepts like Modularization, Use Case Analysis, Sequence Diagrams, and the MVC pattern. Those concepts come together to form the backbone of real-world software development. In the field, software engineering isn't just about writing code; it's about creating systems that are scalable, maintainable, and adaptable to changing requirements. Whether you're building a small application or a large-scale system, these principles guide how you design, implement, and manage your software.

By understanding how to structure your code and system architecture thoughtfully, you ensure your projects can evolve over time, handle complexity, and meet both business and user needs. This knowledge is what separates a developer from a software engineer, and mastering these skills will set you up for long-term success in your careers.

CoGrammar
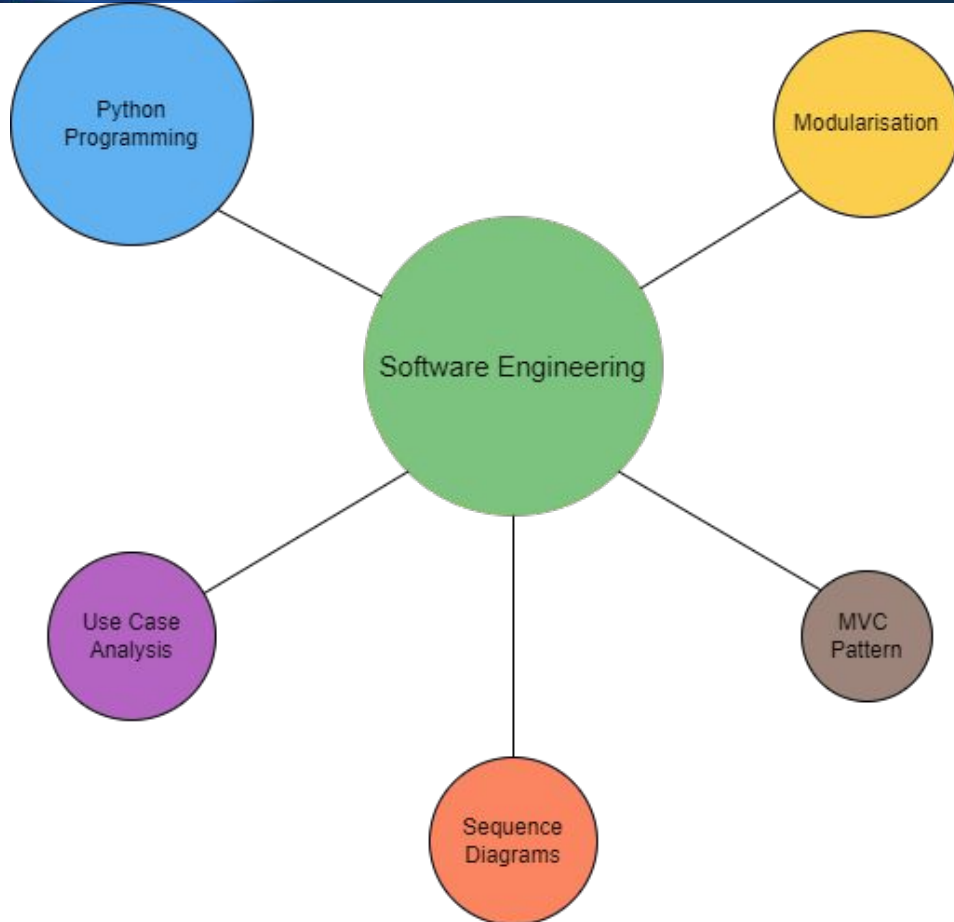
Building the Foundation: Essential Software Engineering Concepts
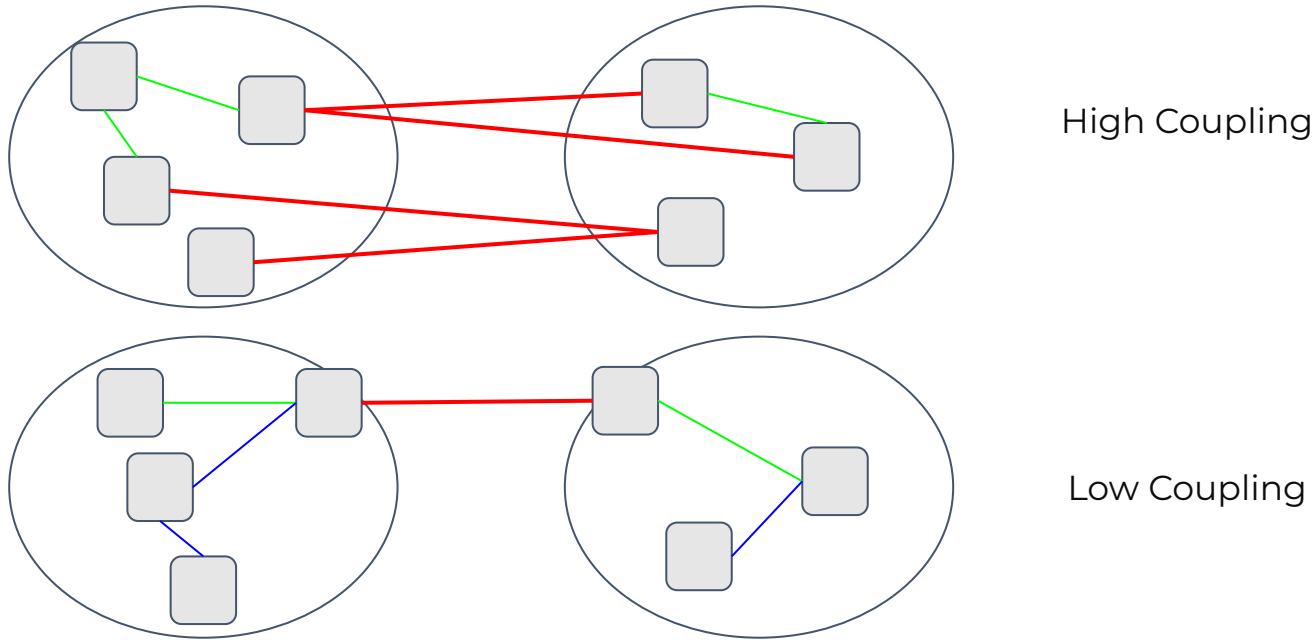
Modularization & Separation of Concerns

- What is …
  - **Modularization**: Breaking a system into manageable, independent modules.
  - **Separation of Concerns**: Ensuring that different parts of a system are responsible for distinct concerns.
- Why does it matter?
  - Promotes maintainability, reduces complexity.
  - Makes code reusable and testable.
  - Enhances scalability of the code

CoGrammar

Best Practices for Modularization

- **Design Techniques**:

  - Object-Oriented Programming (OOP)

  - Functional Programming

  - Microservices

- **Key Principles**:

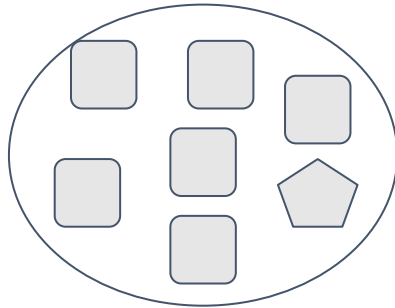 - High cohesion, low coupling.

  - Encapsulation.

CoGrammar

High Coupling

Low Coupling

High Cohesion

Low Cohesion

CoGrammar

- **Example**: Modularized Cafe Operations
  - **Modules**:
    - **Cashier:** Handles customer interactions, orders, and payments.
    - **Barista:** Prepares drinks.
    - **Kitchen:** Prepares food.
  - **Benefits**:
    - **Scalability:** Easily add more staff.
    - **Maintainability:** Fix issues without affecting the whole cafe.
    - **Reusability:** Use payment module in other parts of the business.

CoGrammar

Monolithic vs. Microservices Architecture

Monolithic Architecture
- User Interface
- Business Layer
- Data Interface

Microservices Architecture
- Microservice UI
- Microservice
- Microservice
- Microservice
- Microservice

CoGrammar
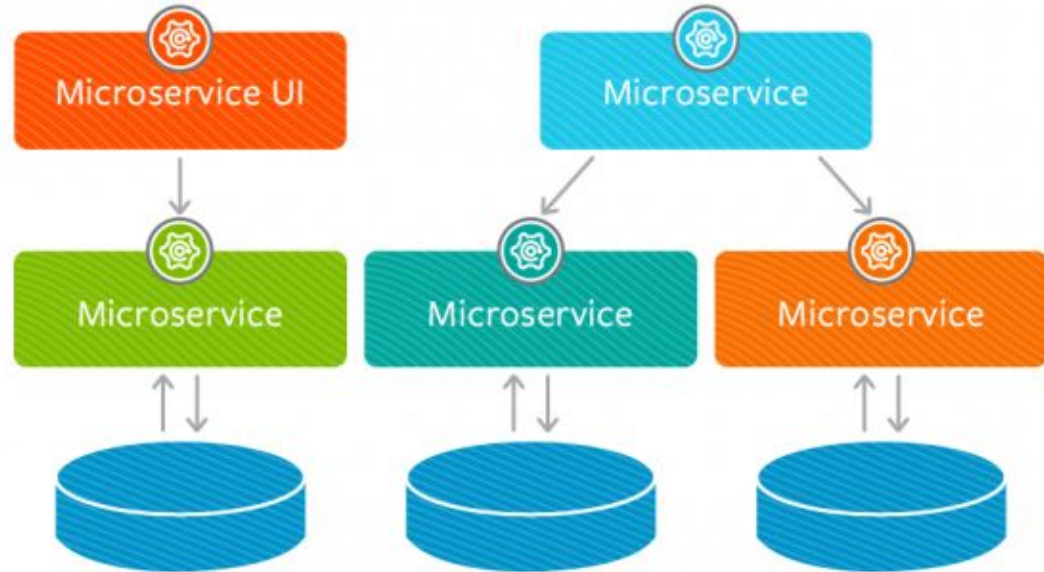
Sequence Diagrams and Use Case Analysis

CoGrammar

Sequence Diagrams

- **What is a Sequence Diagram?**
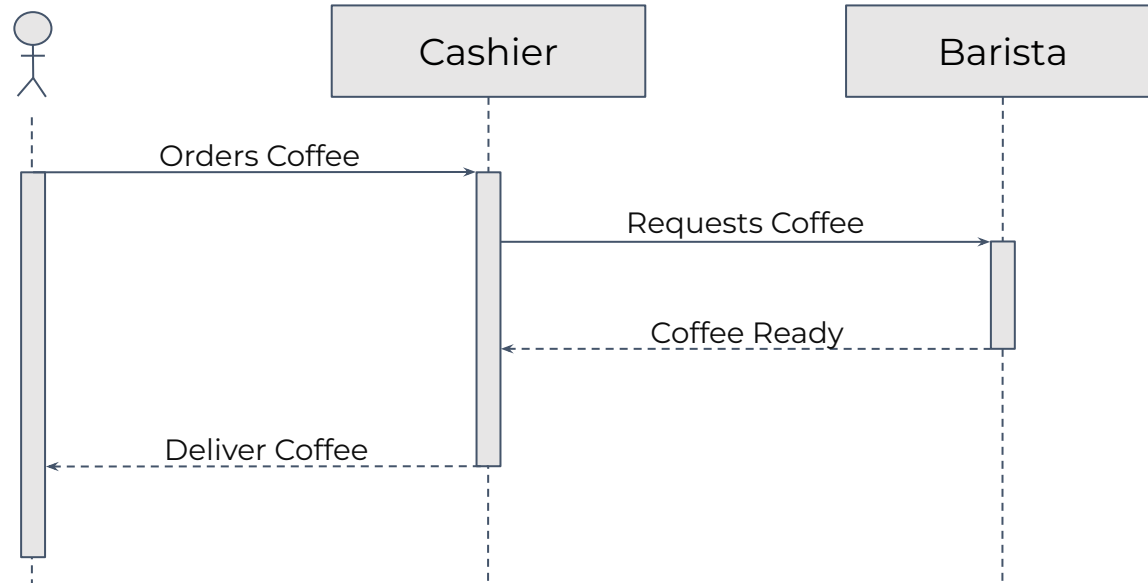 - Visualizes **interactions** over time between objects in the system.
- **Purpose:**
  - Shows how the system handles specific use cases.
- **Elements of Sequence Diagrams:**
  - Lifelines (------), Messages , Actors

CoGrammar

Sequence Diagrams

Use Case Analysis

- **What is a Use Case?**
    - Describes interactions between users and the system.
    - Provides an understanding user requirements and system interactions
- **Importance:**
    - Helps capture functional requirements clearly.
- **Steps:**
    - Identify actors and their goals
    - Create use case diagrams

CoGrammar

Use Case Analysis

Café System

- Customer
- Cashier
- Barista

- Orders Coffee
- Processes Order
- Prepares Coffee
- Collects Payment

CoGrammar

Model-View-Controller Pattern

CoGrammar

Model-View-Controller Pattern

- **MVC Components:**
  - **Model:** Represents the data and logic
  - **View:** Displays the user interface
  - **Controller:** Handles user input and updates the model and view
- **Benefits:**
  - Improved maintainability, testability, scalability
  - Real-world Applications:
  - Web and mobile development frameworks (e.g., Django, React)

CoGrammar

Model-View-Controller Pattern
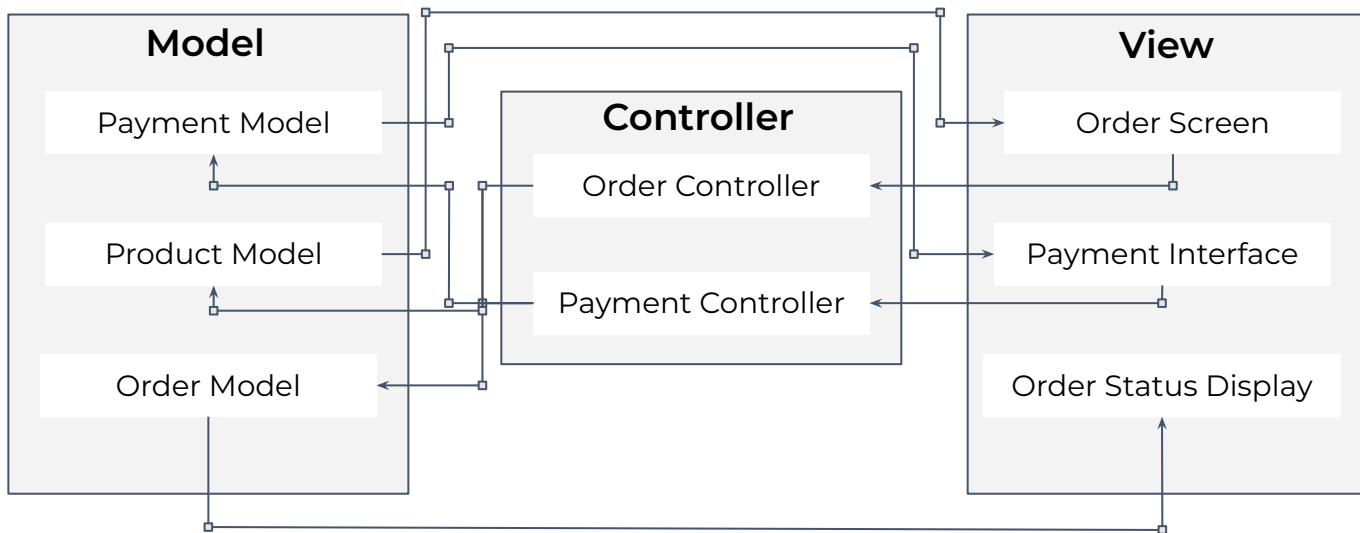
Class Diagrams and CRUD Matrices

Class Diagrams and CRUD Matrices

- **Class Diagrams:**
  - **Purpose:** Representing the structure of a software system
- **Components:**
  - Classes, attributes, methods, relationships
- **CRUD Matrices:**
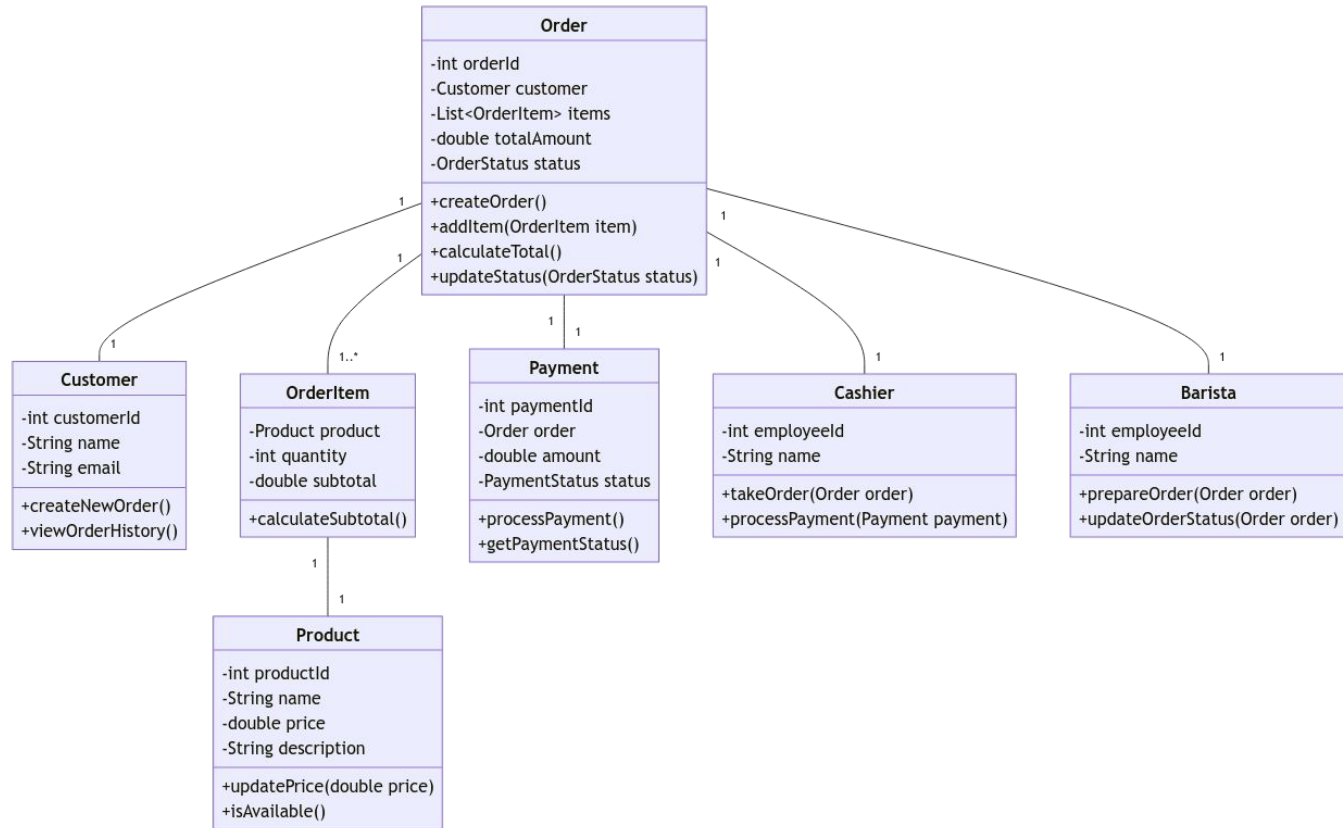 - **Purpose:** Identifying the operations (Create, Read, Update, Delete) needed for each entity
- **Components:**
  - Entities, operations

CoGrammar

Order

-int orderId
-Customer customer
-List<OrderItem> items
-double totalAmount
-OrderStatus status

+createOrder()
+addItem(OrderItem item)
+calculateTotal()
+updateStatus(OrderStatus status)

Customer

-int customerId
-String name
-String email

+createNewOrder()
+viewOrderHistory()

OrderItem

-Product product
-int quantity
-double subtotal

+calculateSubtotal()

Payment

-int paymentId
-Order order
-double amount
-PaymentStatus status

+processPayment()
+getPaymentStatus()

Cashier

-int employeeId
-String name

+takeOrder(Order order)
+processPayment(Payment payment)

Barista

-int employeeId
-String name

+prepareOrder(Order order)
+updateOrderStatus(Order order)

Product

-int productId
-String name
-double price
-String description

+updatePrice(double price)
+isAvailable()

Summary

- **Modularization:** Breaking down software into independent modules enhances maintainability, scalability, and reusability.

- **Sequence Diagrams:** Visualising component interactions elucidates system behaviour over time, aiding in comprehension and optimization.

- **Use Case Diagrams:** Representing system functionalities from user viewpoints aids in requirement analysis and stakeholder communication.

- **Separation of Concerns:** Emphasizes dividing a software system into distinct sections, each responsible for a separate concern or aspect of functionality.

- **MVC:** software architectural pattern widely used in web development. It divides an application into three interconnected components: Model (data management), View (user interface), and Controller (business logic).

- **Class diagrams:** are tools used in software development to plan and organize class functionalities.

- **CRUD matrices:** tools used in software development to plan and organize class functionalities.

CoGrammar

Sequence Diagrams

Questions and Answers

CoGrammar

Let's take a short break

CoGrammar

Thank you for attending

SKILLS
FOR LIFE
SKILLS BOOTCAMPS

Department
for Education

CoGrammar