Week 5: Software Design

Dr. Sridhar Iyer, IIT Bombay
Dr. Prajish Prasad, FLAME University

Introduction to Software Design

Dr. Sridhar Iyer, IIT Bombay
Dr. Prajish Prasad, FLAME University

Outcome of the Design Process

- Components
- Interfaces among different components
- Data structures of individual components
- Algorithms required to implement individual components

Outcome of the Design Process

- Components
  - Modules - collection of functions and data
  - Should accomplish some well-defined tasks

Outcome of the Design Process

- Components
  - Modules - collection of functions and data
  - Should accomplish some well-defined tasks
- Interfaces
  - How components communicate with each other

Outcome of the Design Process

- Components
  - Modules - collection of functions and data
  - Should accomplish some well-defined tasks
- Interfaces
  - How components communicate with each other
- Data structures of individual components
  - Suitable data structures for storing and managing data

Outcome of the Design Process

- Components
  - Modules - collection of functions and data
  - Should accomplish some well-defined tasks
- Interfaces
  - How components communicate with each other
- Data structures of individual components
  - Suitable data structures for storing and managing data
- Algorithms required to implement individual components

Outcome of the Design Process

- **Components**
  - Modules - collection of functions and data
  - Should accomplish some well-defined tasks
- **Interfaces**
  - How components communicate with each other
- Data structures of individual components
  - Suitable data structures for storing and managing data
- Algorithms required to implement individual components

Software Architecture -
High Level Software Design

Seller Portal User Stories

Feature: View inventory

As an **independent seller,**

I want to **view my inventory**

So that **I can take stock of products which are low in number**

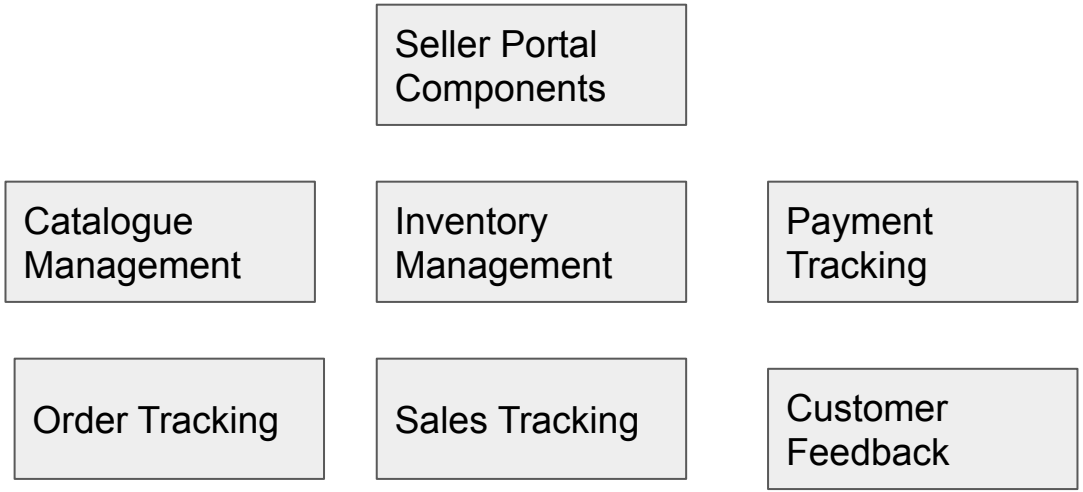
Feature: Track customer feedback

As an **independent seller,**

I want to **view my customers' feedback for each product**

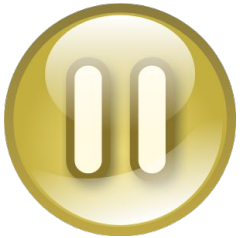
So that **I can get a sense of pertinent issues in my products**

Seller Portal Components



Reflection Spot

What is a good software design?
You have come up with the main components and interfaces of your system. How can you tell that is good?



Please pause the video and written down your responses

Characterizing a Good Software Design

- Correctness: Correctly implement all the functionalities of the system

Characterizing a Good Software Design

- Correctness: Correctly implement all the functionalities of the system
- Efficiency: Ensure that resources - time, space, cost are managed well

Characterizing a Good Software Design

- Correctness: Correctly implement all the functionalities of the system
- Efficiency: Ensure that resources - time, space, cost are managed well
- Maintainability: Easy to change

Characterizing a Good Software Design

- Correctness: Correctly implement all the functionalities of the system
- Efficiency: Ensure that resources - time, space, cost are managed well
- Maintainability: Easy to change
- Understandable: by everyone in the development team

Design Modularity

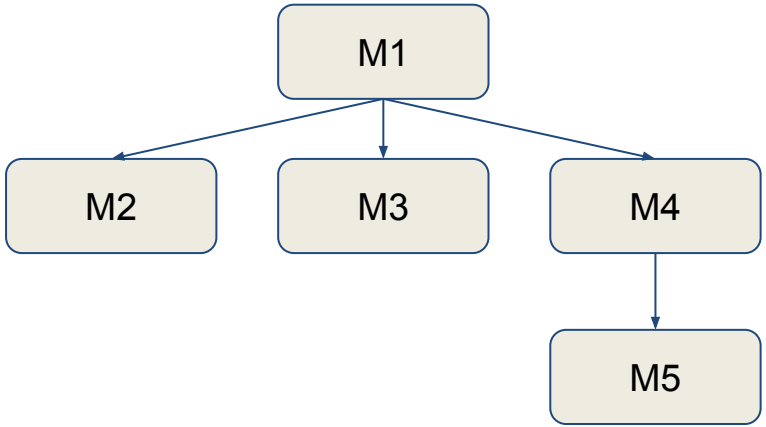
Dr. Sridhar Iyer, IIT Bombay
Dr. Prajish Prasad, FLAME University

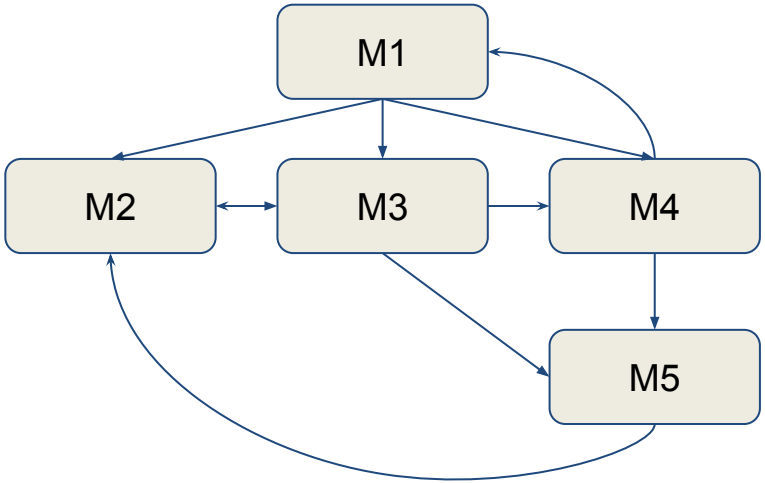
Characterizing a Good Software Design

- Correctness: Correctly implement all the functionalities of the system
- Efficiency: Ensure that resources - time, space, cost are managed well
- Maintainability: Easy to change
- Understandable: by everyone in the development team

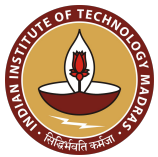
Reflection Spot

Which design is better, easy to understand? Why?



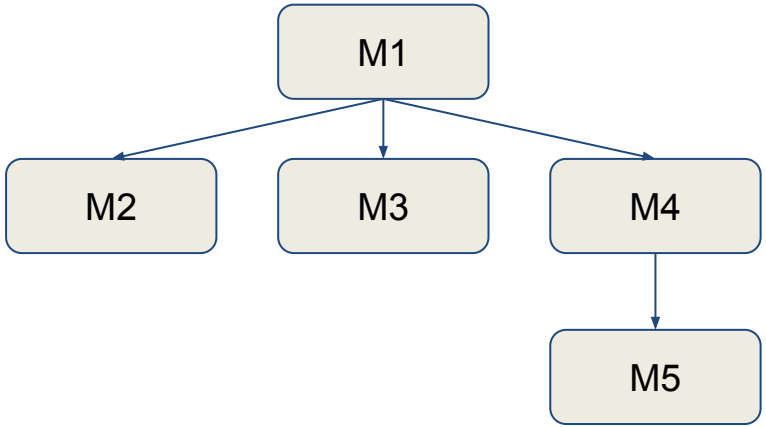


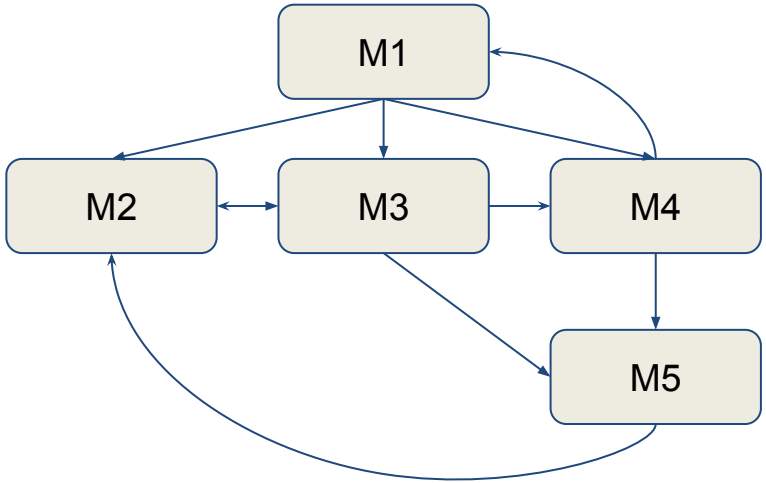
Please pause the video and written down your responses



Answer to Reflection Spot

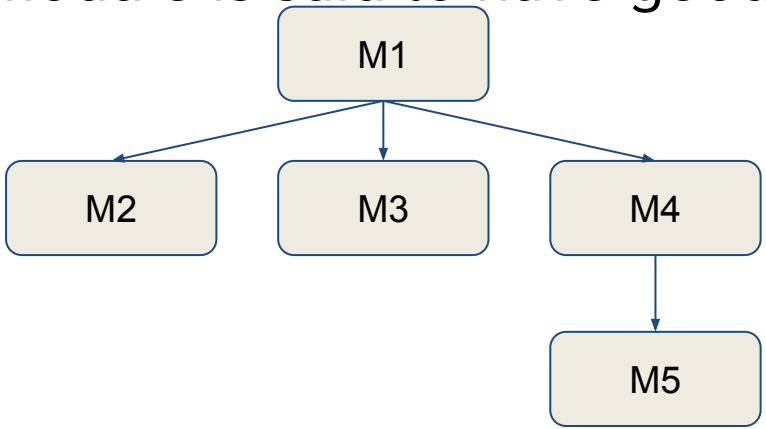
Highly coupled modules share a lot of data among themselves.

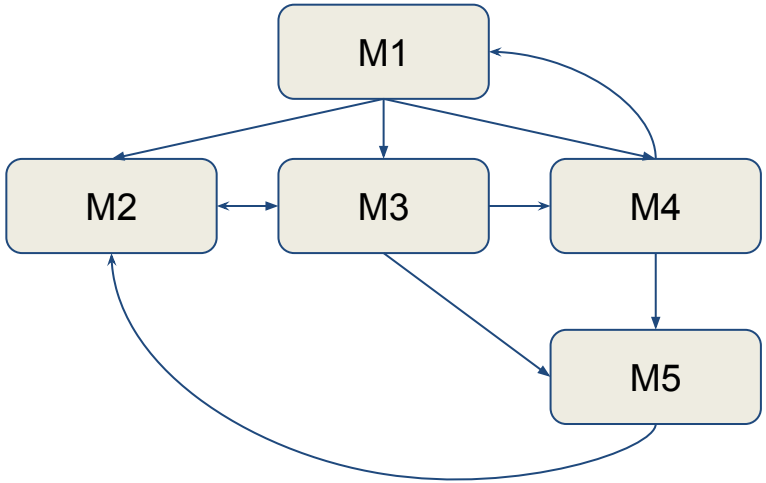




Answer to Reflection Spot

When all functions in a module perform a single objective, the module is said to have good cohesion





Summary: Coupling and Cohesion

- Coupling - measure of the degree of interaction between two modules

- Cohesion - measure of how functions in a module cooperate together to perform a single objective

Reflection Spot

What are characteristics of a good design?

A. High cohesion, High coupling
B. Low cohesion, High coupling
C. High cohesion, Low coupling
D. Low cohesion, Low coupling

Please pause the video and written down your responses

Low Cohesion and High Coupling

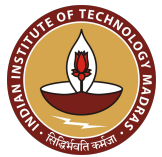
- Low cohesion - functions in each module are performing varied operations

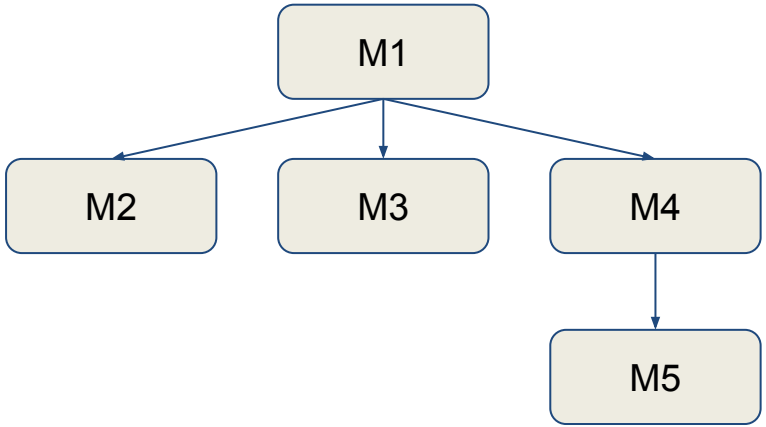
- High coupling – a lot of data is interchanged/shared between modules.

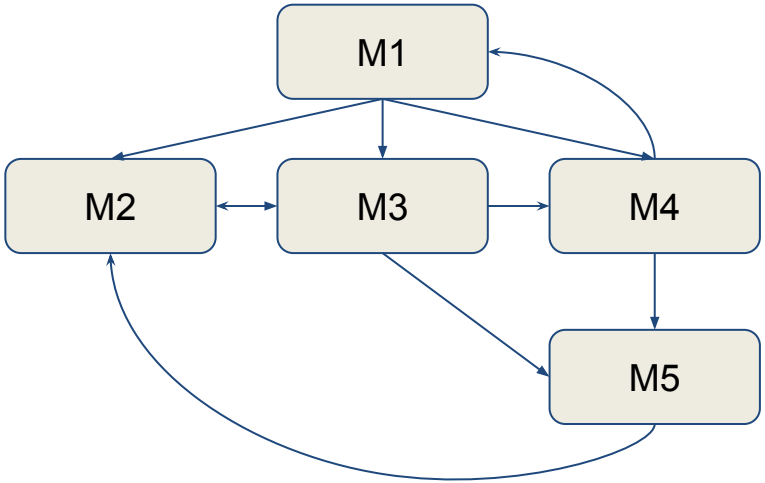
Answer to Reflection Spot

What are characteristics of a good design?

A. High cohesion, High coupling
B. Low cohesion, High coupling
C. High cohesion, Low coupling
D. Low cohesion, Low coupling




M1
M2
M3
M4
M5


M1
M2
M3
M4
M5

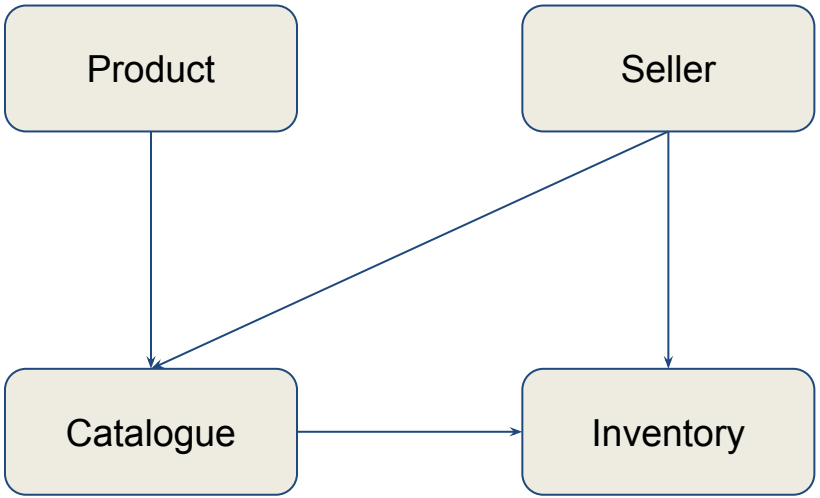
Modular Design

- **Modular:** Problem has been decomposed into a set of modules that have only limited interactions with each other

- **High Cohesion**: Functions of the module co-operate with each other for performing a single objective

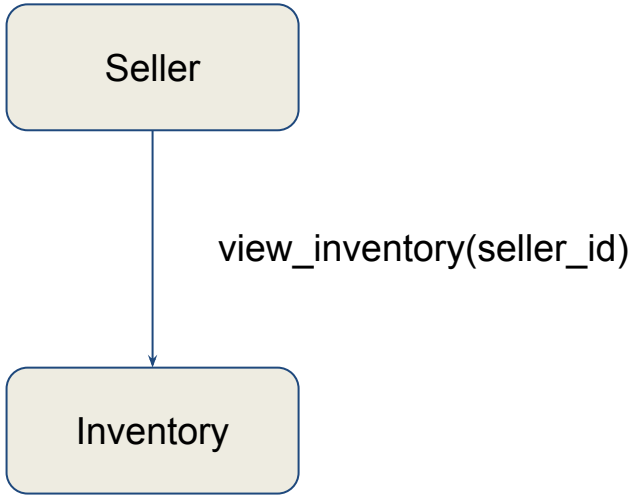
- **High Coupling**: Function calls between two modules involve passing large chunks of shared data

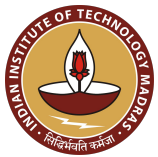
Example: Amazon Seller Portal



Data Coupling

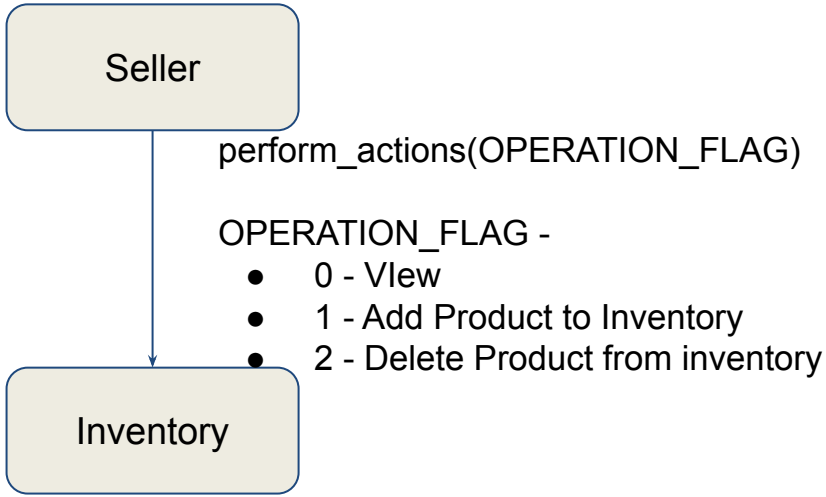
- Communication using a primitive data type - int, float, etc
- Passed as a parameter between two modules





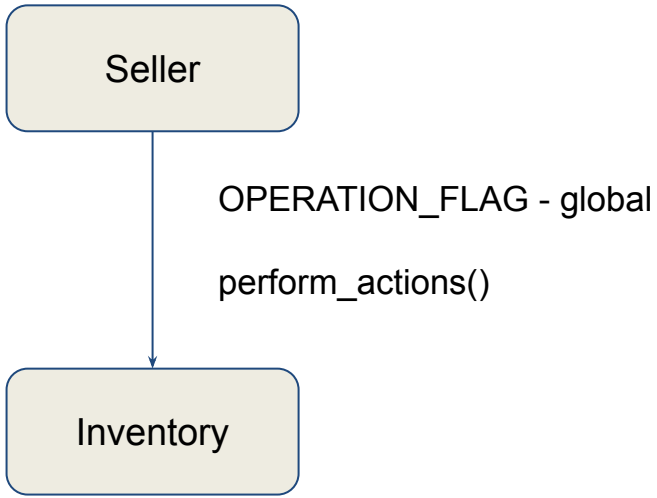
Control Coupling

- Data is passed that influence the internal logic of a module
- E.g. - flags, switches



Common Coupling

- Two modules are commonly coupled, if they share global data items



Content Coupling

- Content coupling occurs between two modules if one refers to the internals of the other module

- Rarely happens in procedural languages or object-oriented systems like Python, Java

Cohesion

When the functions of the module co-operate with each other for performing a single objective, then the module has good cohesion

Functional Cohesion

- Different functions of the module cooperate to complete a single task

- Example – Inventory module
  - view_inventory()
  - add_item_to_inventory()
  - delete_item_inventory()

Sequential Cohesion

- Different functions of the module execute in a sequence
- Output from one function is input to next in the sequence.
- e.g. Order Processing
  - create_order()
  - check_availability()
  - place_order()

Communicational Cohesion

- A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure

- E.g. –
  - Instead of inventory or catalogue, just a product module
  - Add, edit, delete of catalogue and inventory happens in this module itself

Procedural Cohesion

- The activities in the module are related by sequence
- Set of functions in the module are executed one after the other, work towards entirely different purposes
- Seller Portal module –
  - login() ➔ add_product() ➔ add_product_to_catalogue() ➔ add_item_to_inventory() ➔ place_order() ➔ update_inventory() ➔ logout()

Coincidental Cohesion

- Module has functions with meaningless relationships with one another

Summary

- Cohesion and Coupling
- Modular Design - High Cohesion and Low Coupling
- Types of Cohesion, Coupling

Software Engineering

Object Oriented Design: Basic Concepts

Dr. Sridhar Iyer, IIT Bombay
Dr. Prajish Prasad, FLAME University

Modular Designs

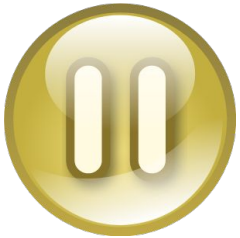
- Modularity - High cohesion, Low coupling
- How do we represent modules?
- This course - Object oriented design approach

Object

- Object - key building block
- Working of a software in terms of interacting objects
- Usually represent a tangible real-world entity

Reflection Spot

What are the key objects you interact with to buy something from a supermarket?



Please pause the video and written down your responses

Answer to Reflection Spot

- Products - Bread, Milk, Shirt
- Shopping Cart
- Create an Order
- Payment - Cash, Credit card, Debit card etc.

What does an object contain

- Data
- Methods
- Encapsulation - Data + Methods
  - Ensures Data Hiding/ Data Abstraction
  - Divides system into parts that have low coupling and high cohesion

Example: Product Object

Private data members:

1. name
2. type
3. size
4. colour
5. brand
6. about
7. manufacturer
8. dimensions
9. cost

Methods

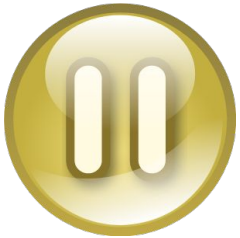
1. add_product()
2. edit_product_details()
3. delete_product()

Class

- Template for object creation
- All objects possessing similar attributes and methods constitute a class
- E.g. Electronic items, clothes, food items - objects of type Product

Reflection Spot

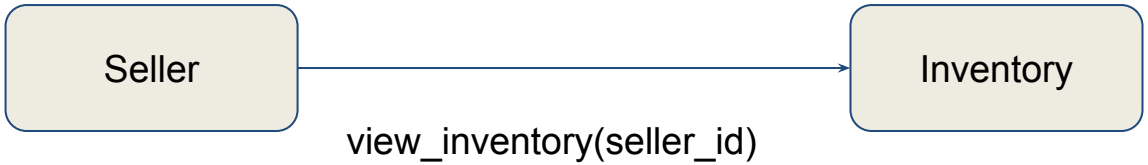
What are ways in which classes can be related to each other?



Please pause the video and written down your responses

Class Relationships: Association

- Take each other's help to perform some functions

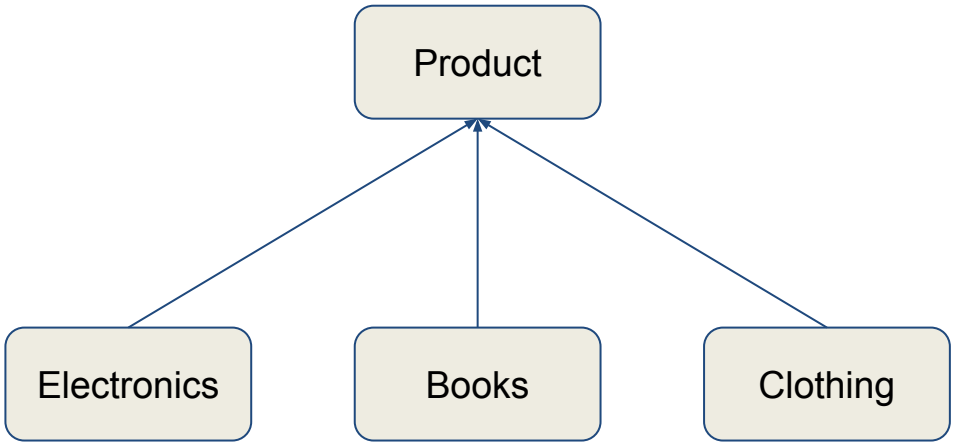


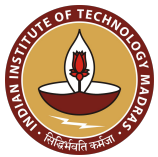
Composition

- Represent whole/part relationships
- E.g. - Catalogue has a list of products

Inheritance

- Extend features of an existing class
- E.g. Product - divide into sub-classes
  - Electronics, Books, Clothing etc.





Dependency

- If Class B depends on Class A, if any changes are made to Class A, changes have to be made to Class B as well

- E.g. - Class B implements an Interface A.
  If A is changed, B also has to be changed

How to identify class relationships

- Association
  - A delegates to B
  - A needs help from B
- Composition/Aggregation
  - B is a part of A
  - A contains B
  - A is a collection of Bs
- Inheritance -
  - A is a kind of B
  - A is a specialisation of B

Summary

- Design - providing structure to a software system
- Outcome of the design process
  - Different modules required
  - Relationships among modules
- Communicating the design to other members is important

Software Engineering

Unified Modelling Language Diagrams

Dr. Sridhar Iyer, IIT Bombay
Dr. Prajish Prasad, FLAME University

Modelling

- Modelling - Creating an external, explicit representation of the system to be built

- Unified Modelling Language (UML) - Help represent the software design via
  - Multiple views
  - Greater level of detail

Important Views of a Software System

- Structural view
- Dynamic behavioural view

Structural View

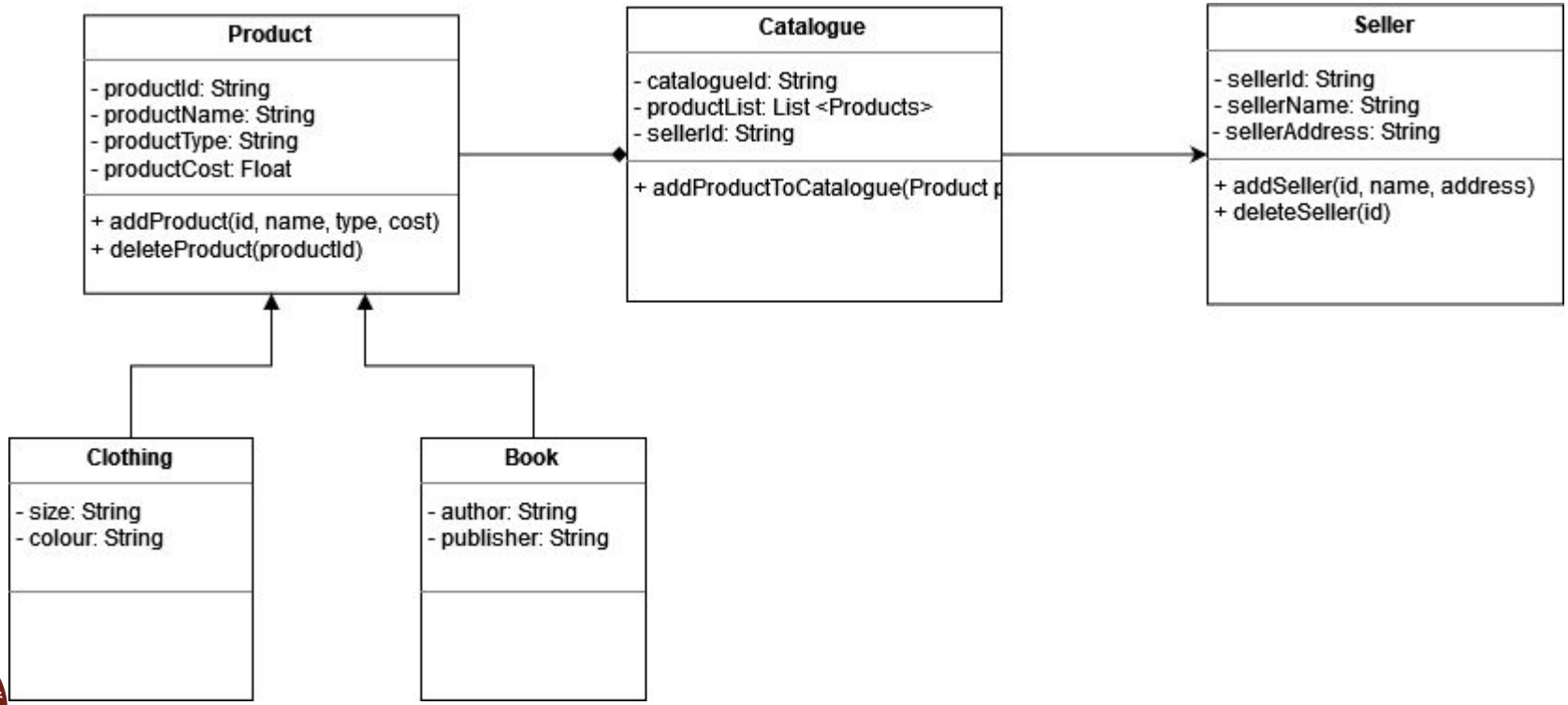
- Structure/components of the software system, and relationships

- Describes logical parts of the system - classes, data and functions

Structural View: Class Diagram

- Describes the structure of the system
- Describes system's classes, attributes, operations, relationships among objects.

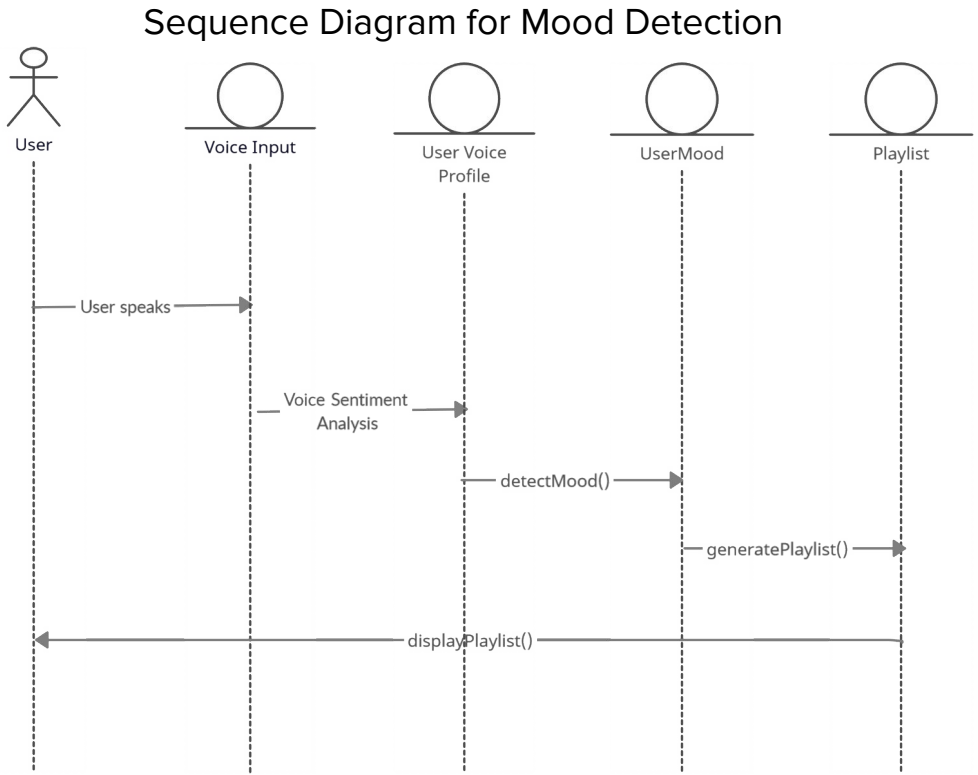
Class Diagram for the Seller Portal



Dynamic Behavioural View

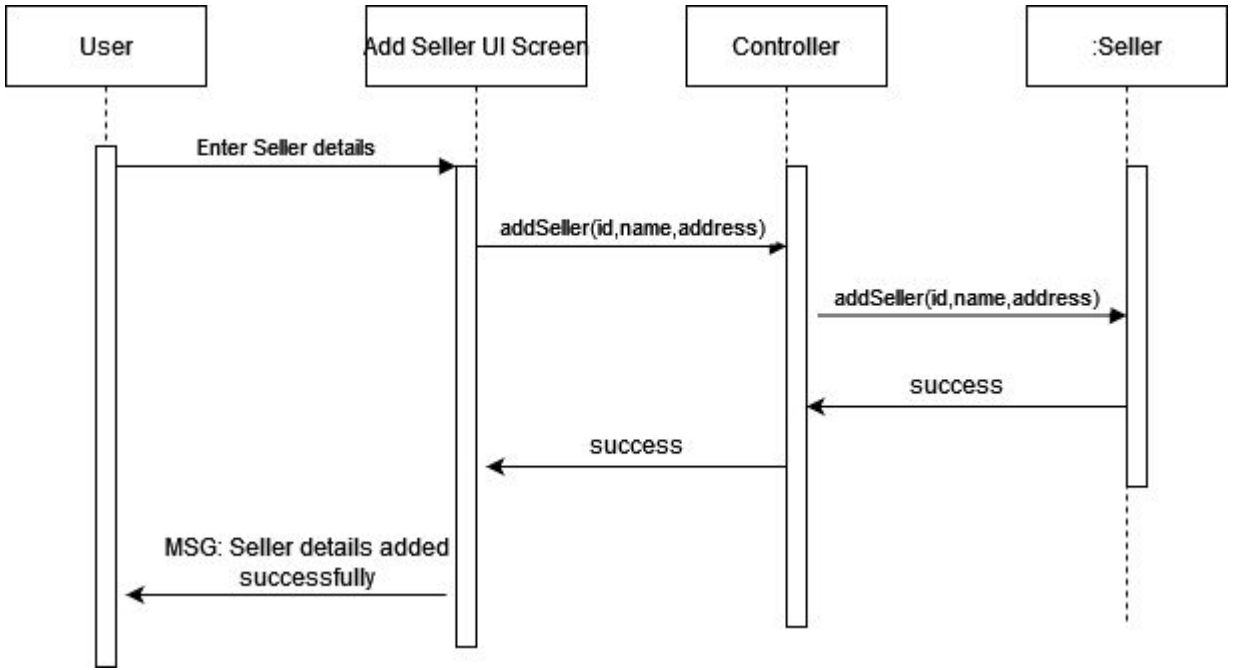
- Describes behaviour of the system over time
- Further classified into:
  - State Machine View - models different states of an object of a class
  - Activity View - models flow of control among computational activities
  - Interaction View - sequence of message exchanges among parts of a system

Behavioural View: Sequence Diagram

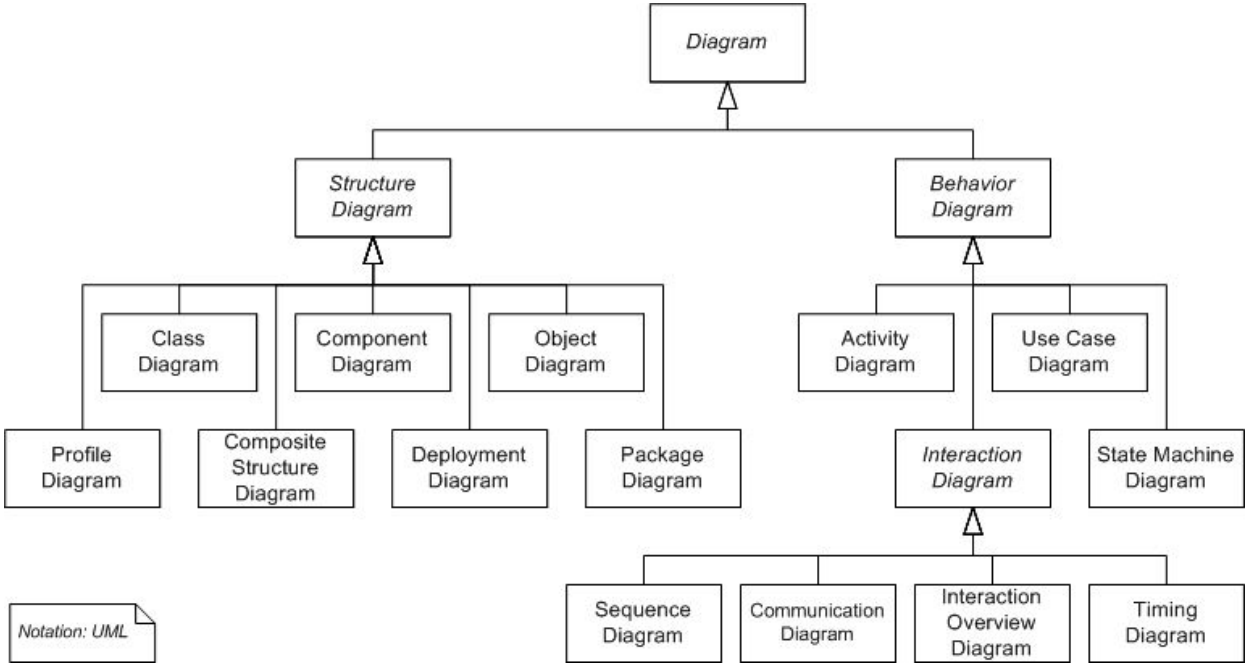


Behavioural View: Sequence Diagram

Sequence Diagram for adding seller



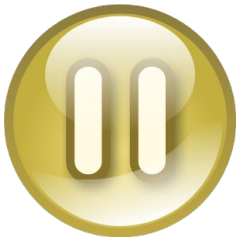
UML Diagrams



Paulo Merson, Public domain, via Wikimedia Commons

Reflection Spot

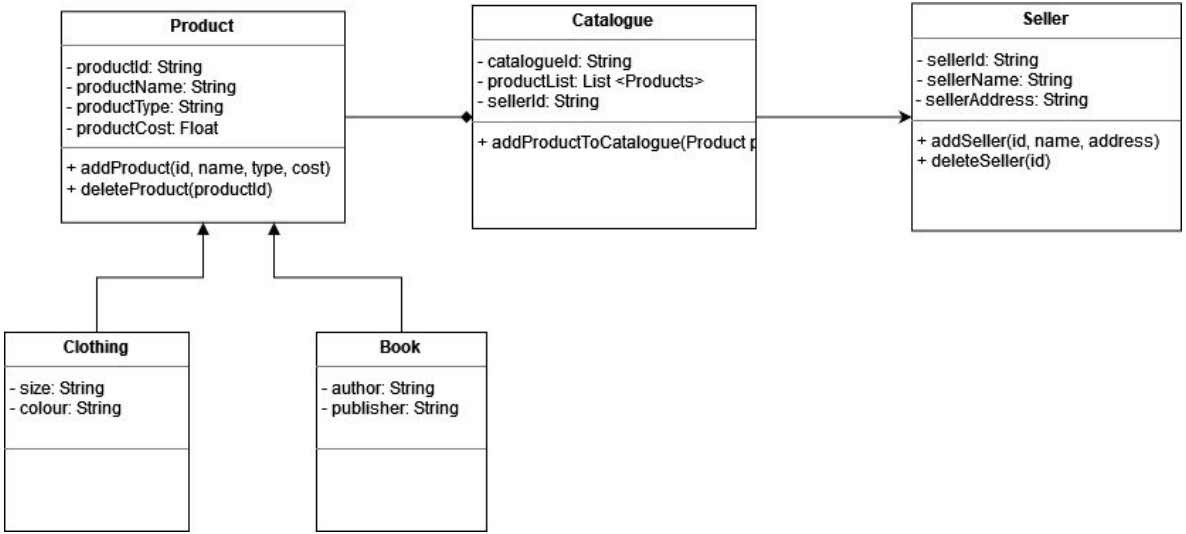
What are different purposes for which UML modelling is used?



Please pause the video and written down your responses

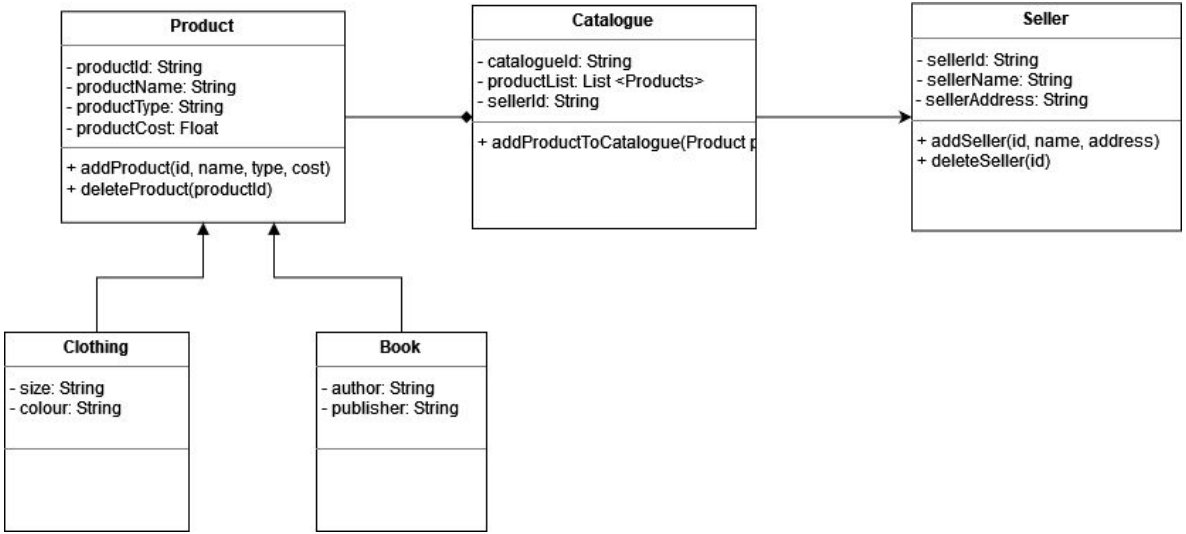
Purposes of Modelling

Vehicle for communication and idea generation



Purposes of Modelling

Guide development of software

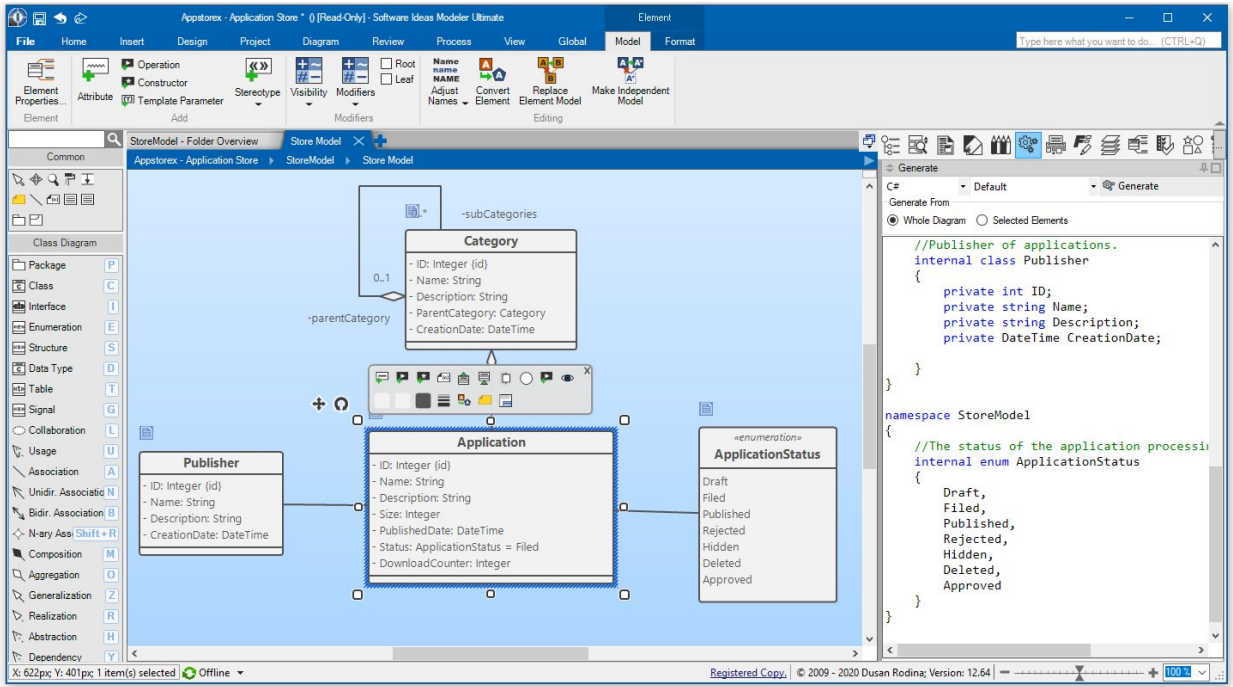


Purposes of Modelling

- Close correspondence with the implementation (code)

- E.g. - Every class in a model ➔ map onto one or more classes in the implementation code

UML to Code



https://www.softwareideas.net/code-generation-diagram-to-code

Purposes of UML Diagrams

- **Vehicle for communication and idea generation**
- Guide development of software
- Generate code from models

VeriSIM: A Learning Environment for Understanding UML Diagrams

Dr. Sridhar Iyer, IIT Bombay
Dr. Prajish Prasad, FLAME University

Unified Modeling Language Diagrams

- UML diagrams - describe different views of the system

- VeriSIM Learning Environment -
  - **Veri**fying designs by **Sim**ulating Scenarios
  - develop an integrated understanding of class and sequence diagrams

Software Design Context in VeriSIM

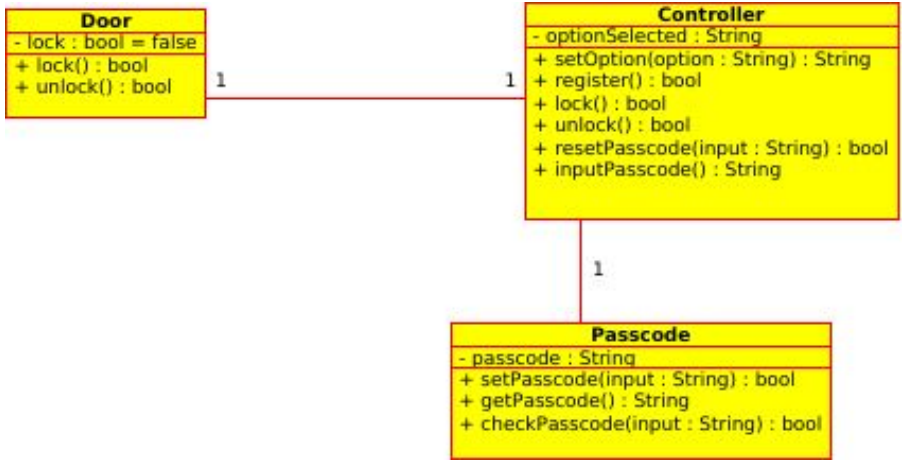
Automated Door Locking System:

Requirements:

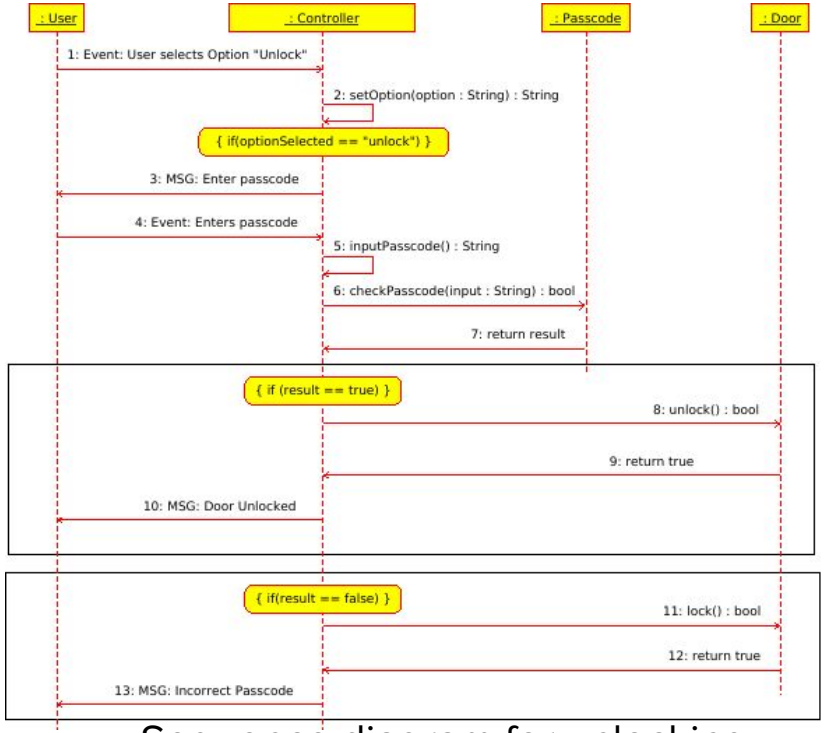
1. If the passcode hasn't been set yet, the user can **register** and enter a required passcode.
2. When the user chooses the **lock option**, and enters the correct passcode, the door should lock. If the passcode is incorrect, the door remains unlocked.
3. When the user chooses the **unlock option**, and enters the correct passcode, the door should unlock. If the passcode is incorrect, the door remains locked.

UML Diagrams

Requirement: When the user chooses the **unlock option**, and enters the correct passcode, the door should unlock. If the passcode is incorrect, the door remains locked.



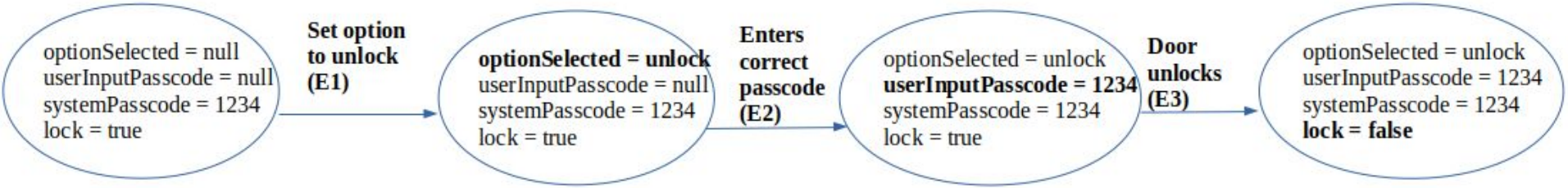
Class diagram



Sequence diagram for unlocking

Design Tracing Strategy in VeriSIM

Scenario:
When the door is initially locked and the user selects the unlock option and enters the correct passcode, the door unlocks”



Construct a state diagram which models the scenario

Activities in VeriSIM