Software Design using the Unified Modeling Language (UML)

Paul Fodor

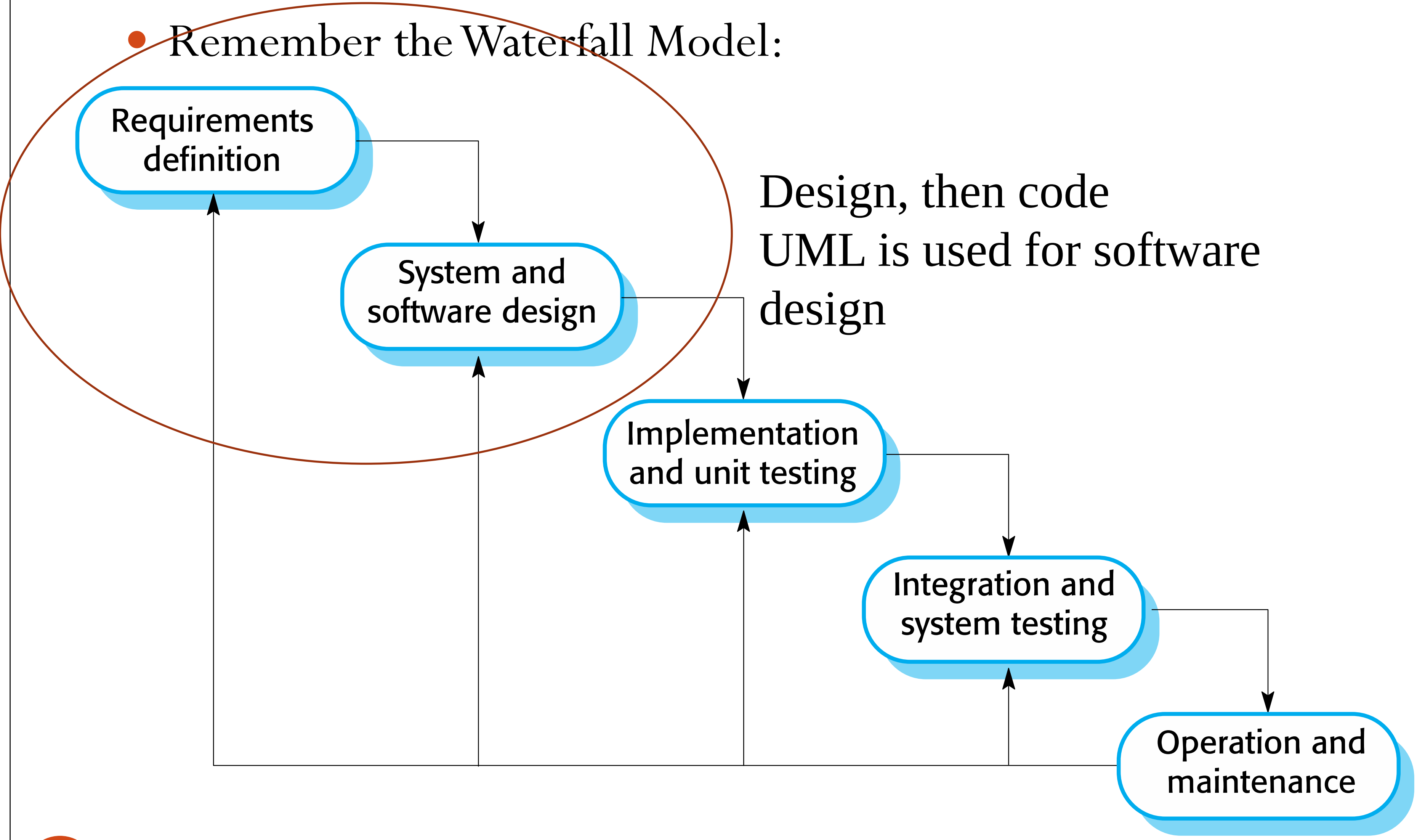
CSE316: Fundamentals of Software Development

Stony Brook University

http://www.cs.stonybrook.edu/~cse316

Software Development Lifecycle

- Remember the Waterfall Model:


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Design, then code

UML is used for software design

Design Principles for Software Engineering

- Software/System Design: design, then code
  - Separate phase in the Software Engineering Software Lifetime:
    - Use basic OO principles like encapsulation and inheritance to make your software more reusable, flexible, easier to maintain.
      - Make sure each of your classes is cohesive: Each class should do ONE THING and do it well
  - Review your design many times before your start coding
    - If a design is bad, then CHANGE IT!
      - Sometimes you might have to scrap all code and restart
        - Don't be afraid to do it. It will save you time and lead to a better implementation.

Unified Modeling Language (UML)

- UML unifies a number of visual design methodologies in software engineering, business modeling and management, database design, and others.
  - UML Class diagrams are a subset of UML that is suitable for conceptual modeling of classes and databases
    - Most used type of UML diagrams
  - UML is also a graphic language for modeling dynamic aspects of a systems behavior
- Because UML is graphic it is particularly appropriate for communicating between the analyst and the customer and between various members of the implementation team

Design Principles for Software Engineering

- We covered UML Class Diagrams with Databases ER diagrams, but there are some details that we should address about design:
  - Design Principles:
    - Encapsulate classes for 2 reasons:
      - show only the simplified public API
        - Classes are about behavior and functionality
      - hide the gory details
    - Code to an Interface!
      - Standardization of interaction for all members of a collection of classes.
    - Never delete functionality from a class because users of that class will not update their way of interacting with that class
      - If you need to change a class, you can add behavior, not remove it

Design

- The Principles of Software Design:
  - OCP – Open to extension, closed to modification
  - DRY – Don't Repeat Yourself
  - SRP – Single Responsibility Principle
  - LSP – Liskov Substitution Principle for OO inheritance

Design

- OCP – Open to extension, closed to modification
  - Once functionality is established, coded, and working, the method should not be changed (closed to modification)
  - Methods should be allowed to be extended (open for extension) to:
    - handle cases where behavior must be different
    - Use subclasses
    - Subclass method can also reuse parent method code by calling it

Design

- DRY – Don't Repeat Yourself
  - Do not have duplicate functionality in different methods or classes (that are not inherited)
  - Move 1 copy of the class to a place where it can be accessed by everyone
  - Improves 'Maintainability'

Design

- SRP – Single Responsibility Principle
  - Every object should have a Single responsibility
  - All contained services should be focused on that responsibility

Design

- LSP – Liskov Substitution Principle
  - Make sure a subclass can be substituted for its parent
  - Methods in subclass with same name should be overwritten (not overloaded)

Barbara Liskov, Professor at MIT, the first women to be granted a doctorate in computer science in the United States and a Turing Award winner who developed the Liskov substitution principle

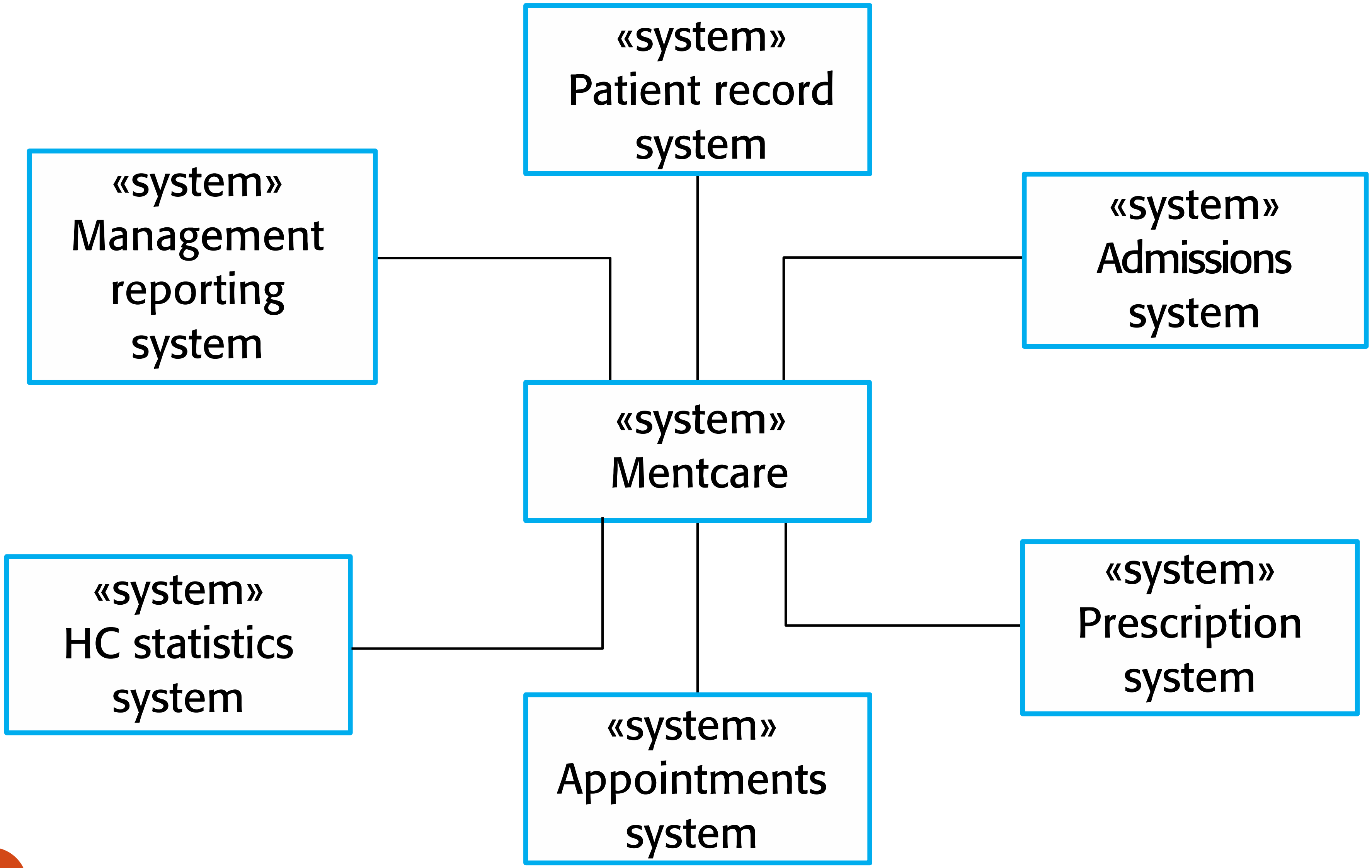
Design Principles for Software Engineering

- There are many consideration in designing a large project:
  - Context of the system with respect to users, boundaries with other systems
    - Process of interaction with externals
  - Structural design of the internals needed for the project
    - Static design: package and class diagrams
    - Dynamic design: interaction with users, with objects, reacting to events

UML Context Models

- Context models illustrate the operational context of a system
  - Architectural model to show the system and its relationship with other systems
  - Also show what lies outside the system boundaries
- System boundaries define what is inside and what is outside the system
  - Show other systems that are used/depend on system being developed
  - Position of the system boundary has a profound effect on the system requirements
  - Defining a system boundary is a political judgment
    - May be pressures to develop system boundaries that increase/decrease the influence/workload of different parts of an organization

UML Context Model Example



«system»
Patient record system

«system»
Management reporting system

«system»
Admissions system

«system»
Mentcare

«system»
HC statistics system

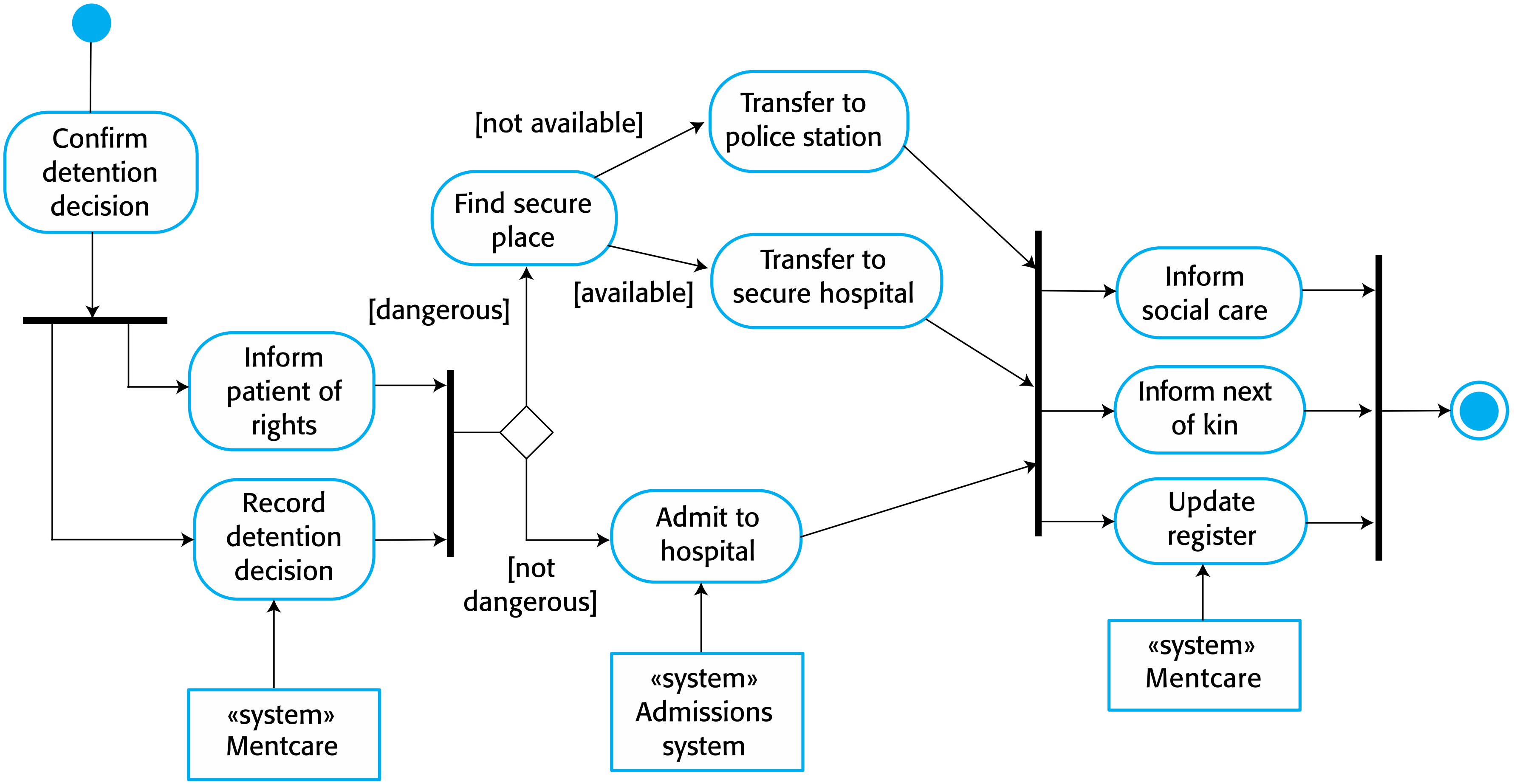
«system»
Prescription system

«system»
Appointments system

Process Perspective

- Context models
  - do not who how the system being developed is used in the environment
- Process models reveal how system being developed is used in broader business processes
  - UML activity diagrams may be used to define business process models

Process Model Example as a UML Activity Diagram

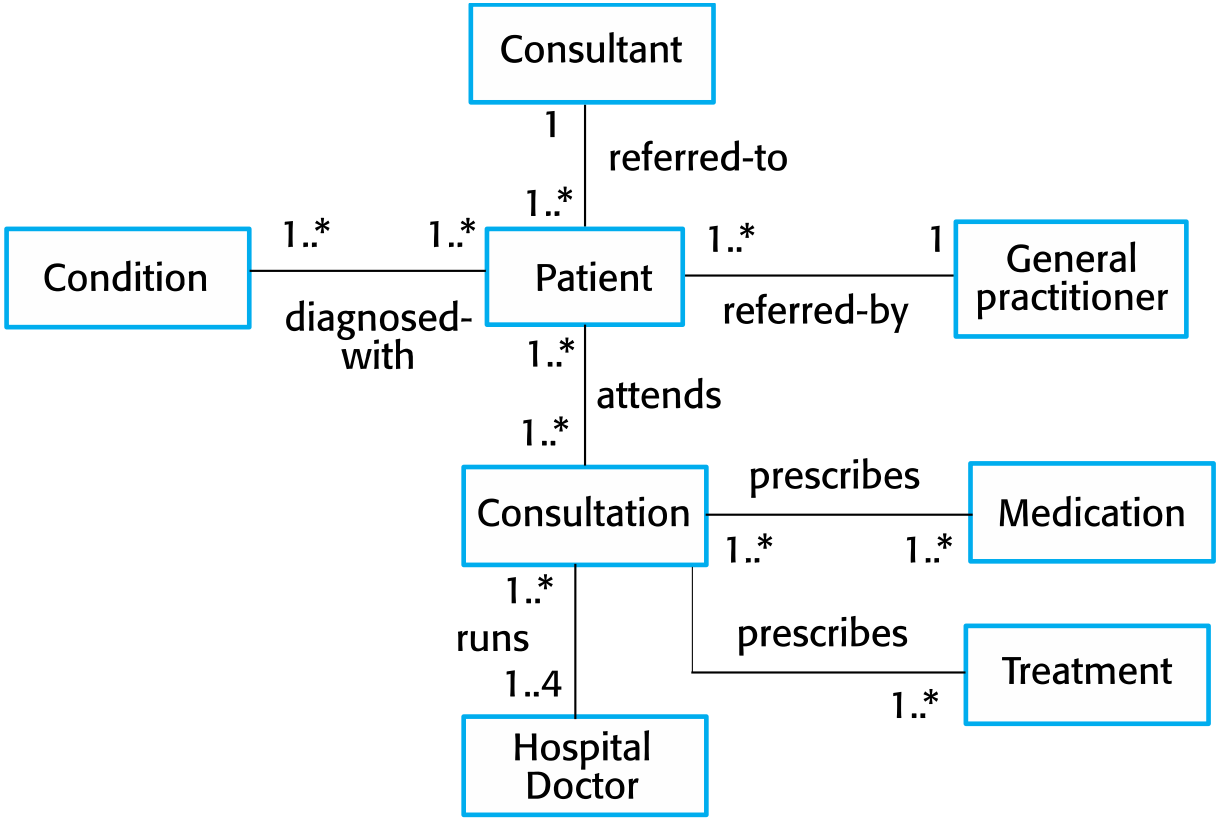


Structural Models

- Structural models of software display organization of a system in terms of the components that make up that system and their relationships
  - Static models ➔ show structure of the system design
  - Dynamic models ➔ show organization of the system when it’s executing
- Structural models of a system are created when discussing and designing system architecture

Static Models

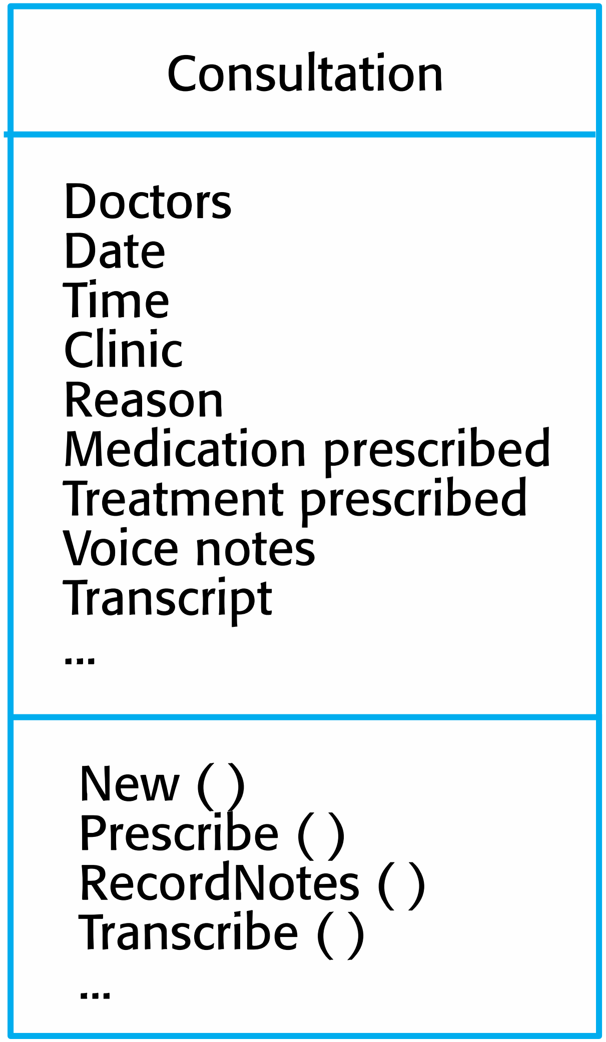
- Class Diagrams:
  - Used when developing an object-oriented system model
    - Shows the classes in a system
    - Shows the associations between these classes



Static Models

- Class Diagram

Details of a class:



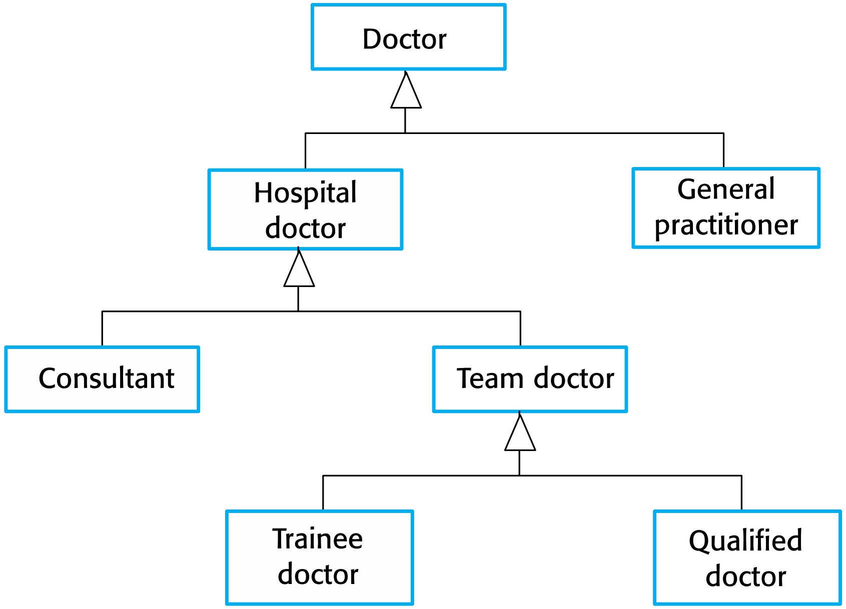
Consultation

Doctors
Date
Time
Clinic
Reason
Medication prescribed
Treatment prescribed
Voice notes
Transcript
...

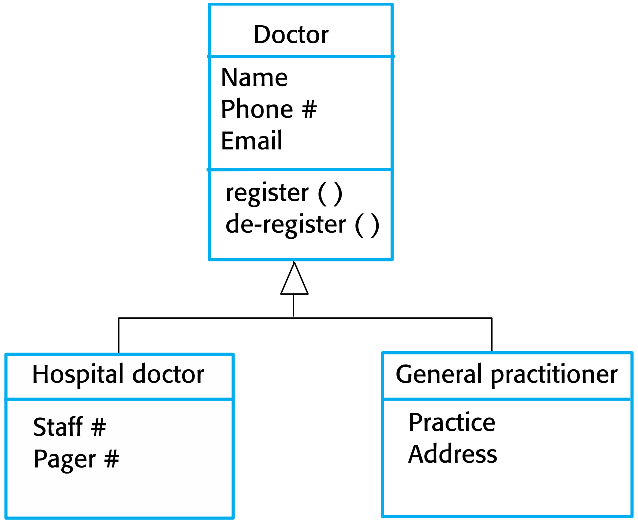
New ( )
Prescribe ( )
RecordNotes ( )
Transcribe ( )
...

Static Models

- Generalization: a technique used to manage complexity:
  - Place the common attributes in more general classes
  - Lower-level classes (subclasses) inherit attributes and operations from their superclasses
  - Lower-level classes then add more specific attributes and operations



or in more detail



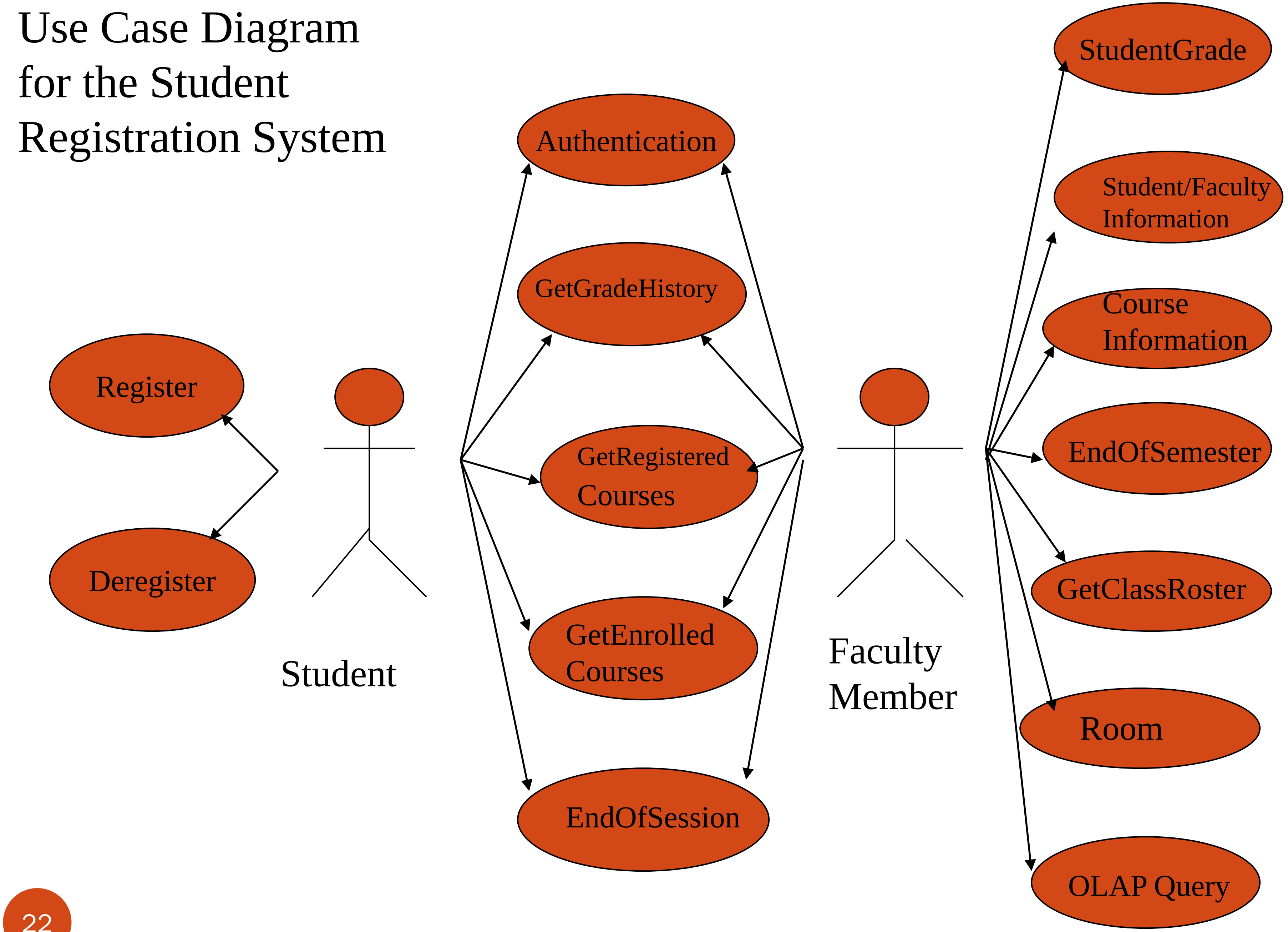
Behavioral Models

- Models of the dynamic behavior of a system as it executes
  - Shows what happens (or what is supposed to happen) when a system responds to stimulus from its environment
- Two types of stimuli:
  - **Data** ➔ Some data arrives that has to be processed by the system
  - **Events** ➔ Some event happens that triggers system processing
    - Events may have associated data but this is not always the case
- Data-driven models can be represented by:
  - Use case diagrams
  - Activity Models
  - Sequence Diagrams

Use Case Diagrams

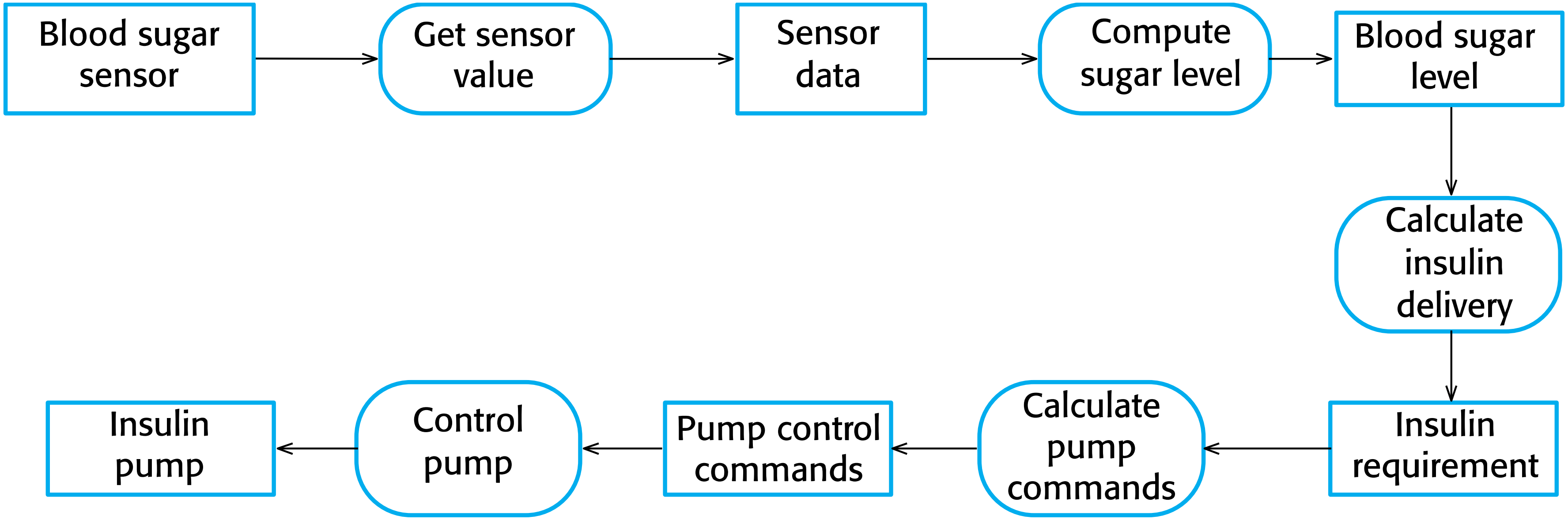
- UML provides a graphic way to display all the use cases in an application
- These diagrams can be used to communicate with the
  - Customer to determine if the current set of use cases is adequate
  - Developers to determine what the system is supposed to do from the customer's viewpoint
- Always included in the Requirements Analysis Specification document in the Waterfall Model

Use Case Diagram for the Student Registration System



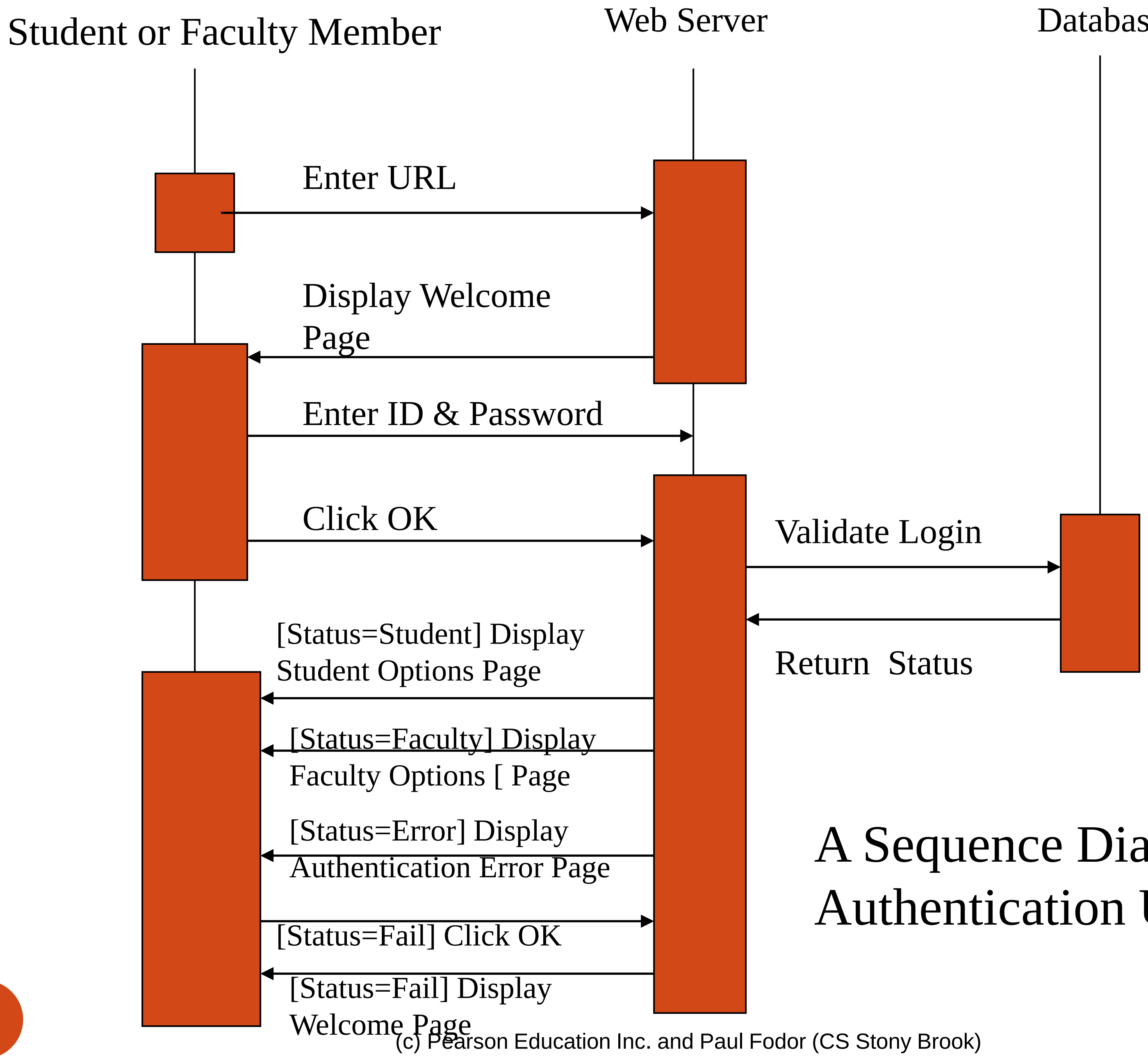
Behavioral Models

- Activity Models
  - An Activity Model of Insulin Pump Operation:



UML Sequence Diagrams

- A graphic display of the temporal ordering of the interactions between the actors in a use case and the other modules in the system
- Sometimes it is part of the plan for preparing the Specification Document to expand each use case into the set of interactions
- It is always part of the Design document in Waterfall model, together with the UML Class diagrams



Student or Faculty Member

Web Server

Database

Enter URL

Display Welcome Page

Enter ID & Password

Click OK

Validate Login

Return Status

[Status=Student] Display Student Options Page

[Status=Faculty] Display Faculty Options [Page

[Status=Error] Display Authentication Error Page

[Status=Fail] Click OK

[Status=Fail] Display Welcome Page

A Sequence Diagram for the Authentication Use Case

Sequence Diagrams

- The actors and pertinent modules are labelled at the top of the diagram
- Time moves downward
- The boxes show when a module or actor is active
- The horizontal lines show the actions taken by the modules or actors
  - Note the notation for conditional actions

    [status=student] Display Student Options Page

Behavioral Models

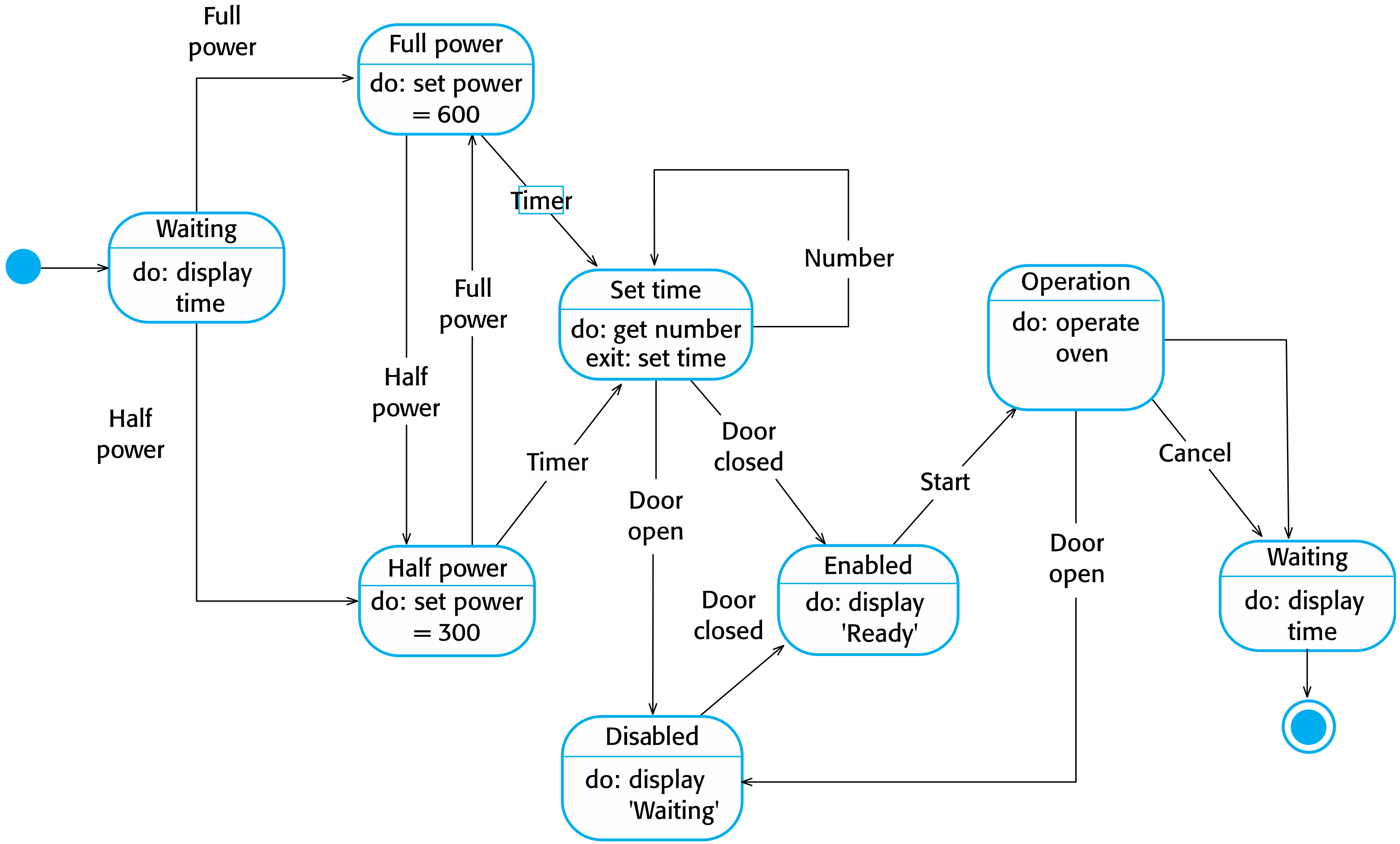
- Event-Driven Modeling:
  - Real-time systems are often event-driven, with minimal data processing.
    - Example: a landline phone switching system responds to events such as 'receiver off hook' by generating a dial tone
  - Event-driven modeling
    - Shows how a system responds to external and internal events
      1. A system usually has a finite number of states
      2. Events (stimuli) may cause a transition from one state to another

Behavioral Models

- State Machine Models:
  - Models the behavior of a system in response to external and internal events
  - Shows system states as nodes and events as arcs between these nodes
    - When an event occurs, system moves from one state to another.
  - UML Statecharts:
    - Used to represent state machine models

Behavioral Models

- UML Statechart Example of a Microwave Oven
  - transition labels are events: like click Full power button



Behavioral Models

- States for Microwave:

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows ‘Half power’.
Full power	The oven power is set to 600 watts. The display shows ‘Full power’.
Set time	The cooking time is set to the user’s input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows ‘Not ready’.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows ‘Ready to cook’.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for five seconds. Oven light is on. Display shows ‘Cooking complete’ while buzzer is sounding.

Behavioral Models

- Stimuli/Events for Microwave:

Stimulus	Description
Half power	The user has pressed the half-power button.
Full power	The user has pressed the full-power button.
Timer	The user has pressed one of the timer buttons.
Number	The user has pressed a numeric key.
Door open	The oven door switch is not closed.
Door closed	The oven door switch is closed.
Start	The user has pressed the Start button.
Cancel	The user has pressed the Cancel button.

Summary

- Models of application systems help us:
  - Understand, present, design and discuss applications with customers and other software developers
- UML diagrams are visual ways to document the requirements and design applications
- Software may be documented from several different perspectives:
  - Conceptual view
  - Process view
  - Development view