Week 3 - Monday

COMP 3100

Last time

- What did we talk about last time?
- Requirements documents

Questions?

More on Requirements

Verifying and validating requirements in traditional processes

- Traditional processes depend on requirements being right
- The following characteristics should be checked:
  - **Clarity:** Are the requirements clear (testable)?
  - **Consistency:** Are there any contradictions?
  - **Completeness:** Is everything covered in sufficient detail?
  - **Correctness:** Do the requirements reflect what stakeholders want?
  - **Well-formedness:** Are the requirements formatted correctly? (Uniquely labeled atomic requirements using "must" and "shall")
- **Reviews** are the process of having various people check the requirements for these characteristics

Verifying and validating requirements in agile processes

- Requirements are always under scrutiny in agile processes
  - Someone is always updating and prioritizing the product backlog
  - Requirements are checked at the end of each sprint
- If the current version of the product behaves incorrectly, it might mean that the requirements are incorrect
- Unfortunately, you need someone who can recognize the errors at the sprint reviews

Requirements management in traditional processes

- Projects start with a product mission statement giving business requirements
- Requirements analysis is the process of gathering stakeholder needs and using them to turn the mission statement into a list of requirements specifications
- The result is a document called a software requirements specification (SRS)
  - This is what you've got to create for Project 1

Book outline for an SRS

1. Introduction
   A. Product Vision
   B. Project Scope
   C. Stakeholders
   D. Design and Implementation Constraints
2. Functional Requirements
   A. Product Behavior
   B. User Interfaces
   C. System Interface
   D. Data Requirements
3. Non-Functional Requirements
4. Other Requirements
5. Glossary

Project 1 outline for SRS

1. Introduction
   A. Purpose of Document
   B. Intended Audience
   C. Scope
   D. Definitions and Terminology
2. Overall Description
   A. Product Functions
   B. User Characteristics
   C. Dependencies
3. Interfaces
   A. User interfaces
   B. Hardware interfaces
   C. Software interfaces
   D. Communications interfaces
4. Functional Requirements
5. Non-functional Requirements

Requirements management in agile processes

- The mission statement or other high-level needs are used to writer big user stories
- Working with stakeholders, the team refines sprintable stories into operational-level and physical-level requirements
- The product owner has the responsibility to update the product backlog as the product evolves

Requirements vs. product design

- Most industries call requirements analysis "product design"
- A lot of other industries design things, but software developers tend not to use the same tools they do
  - Maybe just because we don't call it product design
- It might be more helpful to think about requirements analysis in terms of design
  - Many designs are possible
  - It's smart to come up with several alternatives to see which ones people like best
  - *Designing* is a more active mindset than *gathering requirements*
- Like other problem-solving activities, requirements analysis should involve:
  - Trial and error
  - Iteration
  - Recognition that there isn't a unique solution

Requirements modeling

- When software engineers say *modeling*, they usually mean drawing diagrams
- **Requirements modeling** is making representations (diagrams) that help you understand your requirements
- Both traditional and agile processes use models
- The **Unified Modeling Language (UML)** is the most common set of standards for representing such models
- Some developers use models extensively, and others use them rarely

Kinds of requirements modeling

Model	Show	Typical UML Diagram
Use Case Models	A product interacting with its environment, often **actors** who take on roles	**Use Case Diagram**
Conceptual Models	Relationships between entities	**Class Diagram**
State Diagrams	The states a product can be in and the transitions between those states	**State Diagram**
Decision Trees and Tables	What a product should do under various conditions	**Activity Diagram**
Data Flow Diagrams	How data enters, is processed, and leaves the product	**Activity Diagram or Sequence Diagram**

UML

Modeling

- At both the requirements stage and the design stage, modeling can be useful
- **Modeling** mostly means drawing boxes and arrows
- We want high-level descriptions of:
  - What the thing is supposed to do
  - What parts it's composed of
  - How it does what it does

System modeling

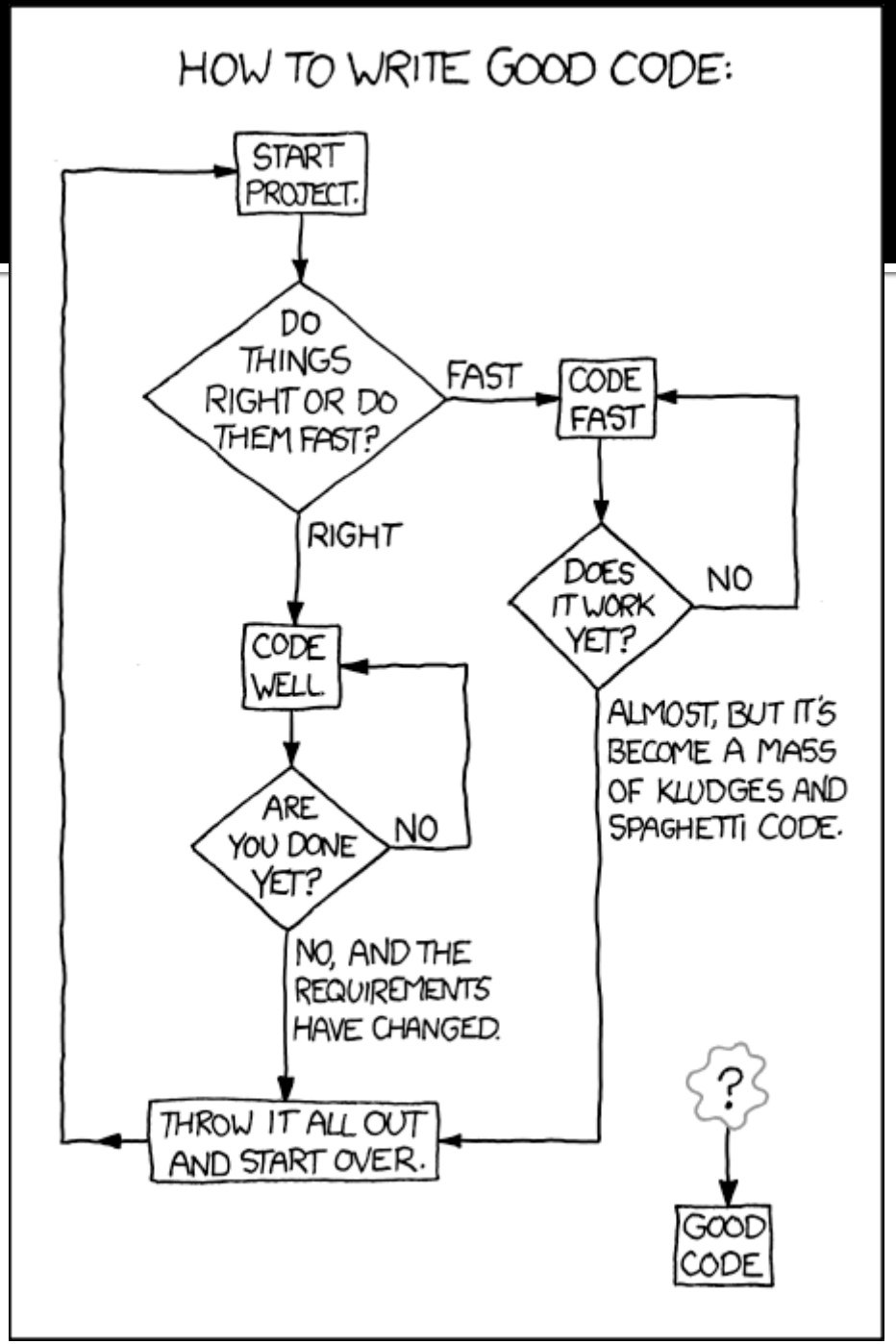
- **Models leave out details**
- Models are useful to help understand a complex system
  - During requirements engineering, models clarify what an existing system does
  - Or models could be used to plan out a new system
- Models can represent different perspectives of a system:
  - **External:** the context of a system
  - **Interaction:** the interactions within the system or between it and the outside
  - **Structural:** organization of a system
  - **Behavior:** how the system responds to events

UML

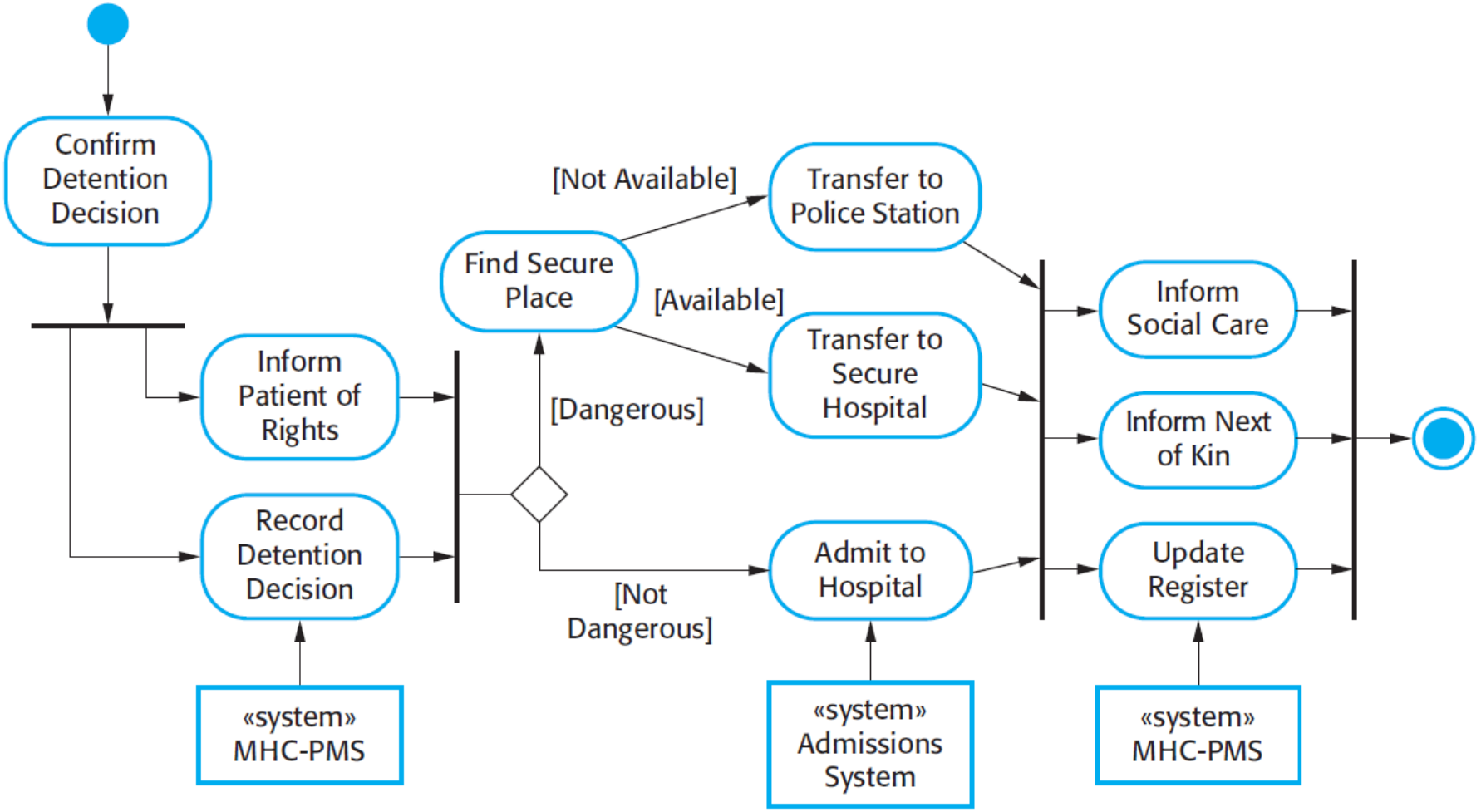
- The **Unified Modeling Language** (UML) is an international standard for graphical models of software systems
- A few useful kinds of diagrams:
  - Activity diagrams
  - Use case diagrams
  - Sequence diagrams
  - State diagrams
- Class diagrams are important enough that we'll talk about them in greater detail

Activity diagrams

- Activity diagrams show the workflow of actions that a system takes
- XKCD of an activity diagram for writing good code
  - From: https://xkcd.com/844/
- Formally:
  - Rounded rectangles represent actions
  - Diamonds represent decisions
  - Bars represent starting or ending concurrent activities
  - A black circle represents the start
  - An encircled black circle represents the end

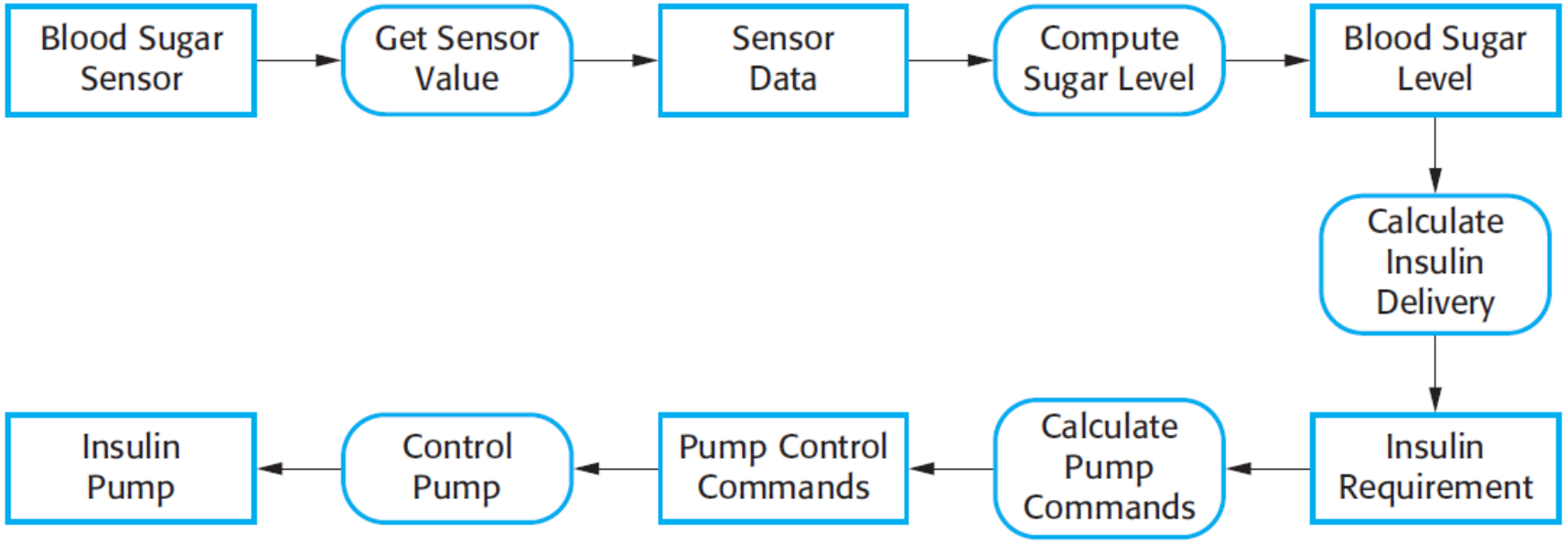


More detailed activity model



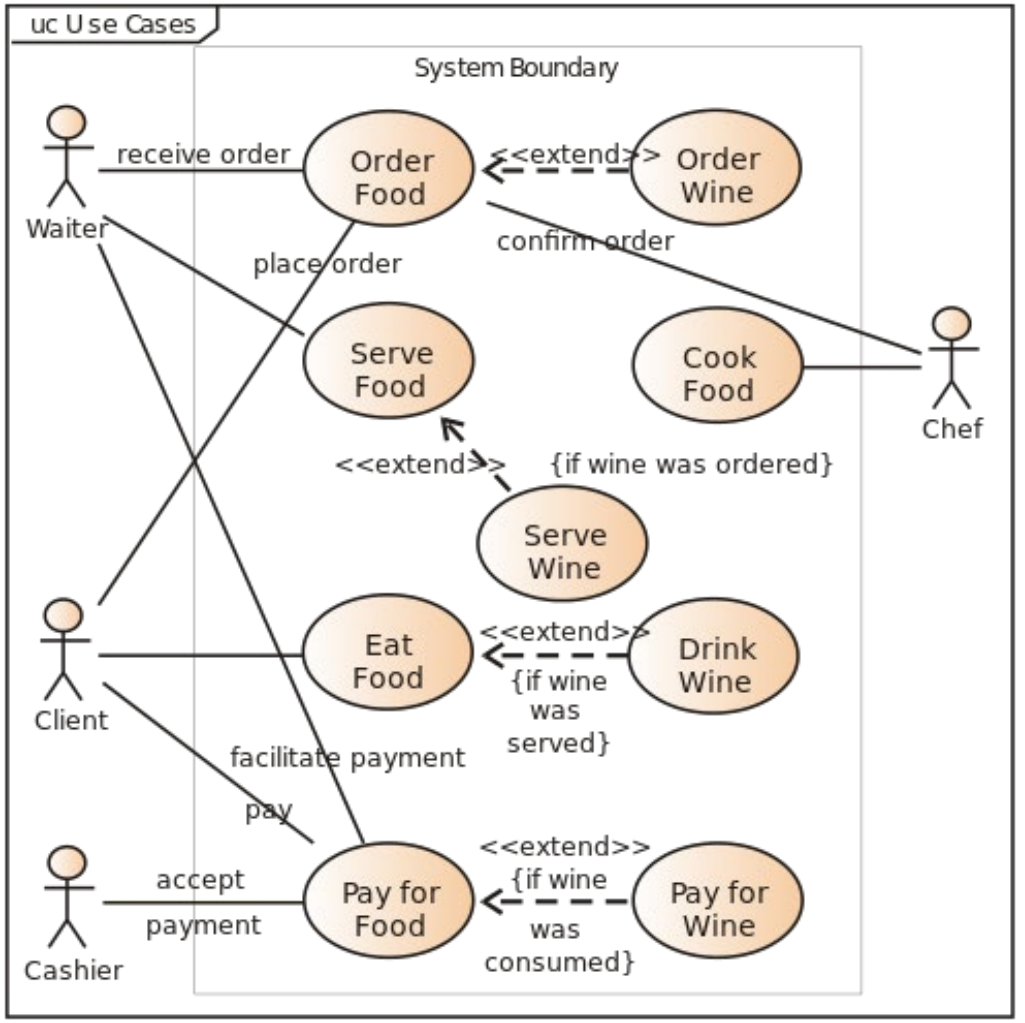
Data-driven modeling

- Data-driven models show how input data is processed to generate output data
- The following is an activity diagram that shows how blood sugar data is processed by a system to deliver the right amount of insulin



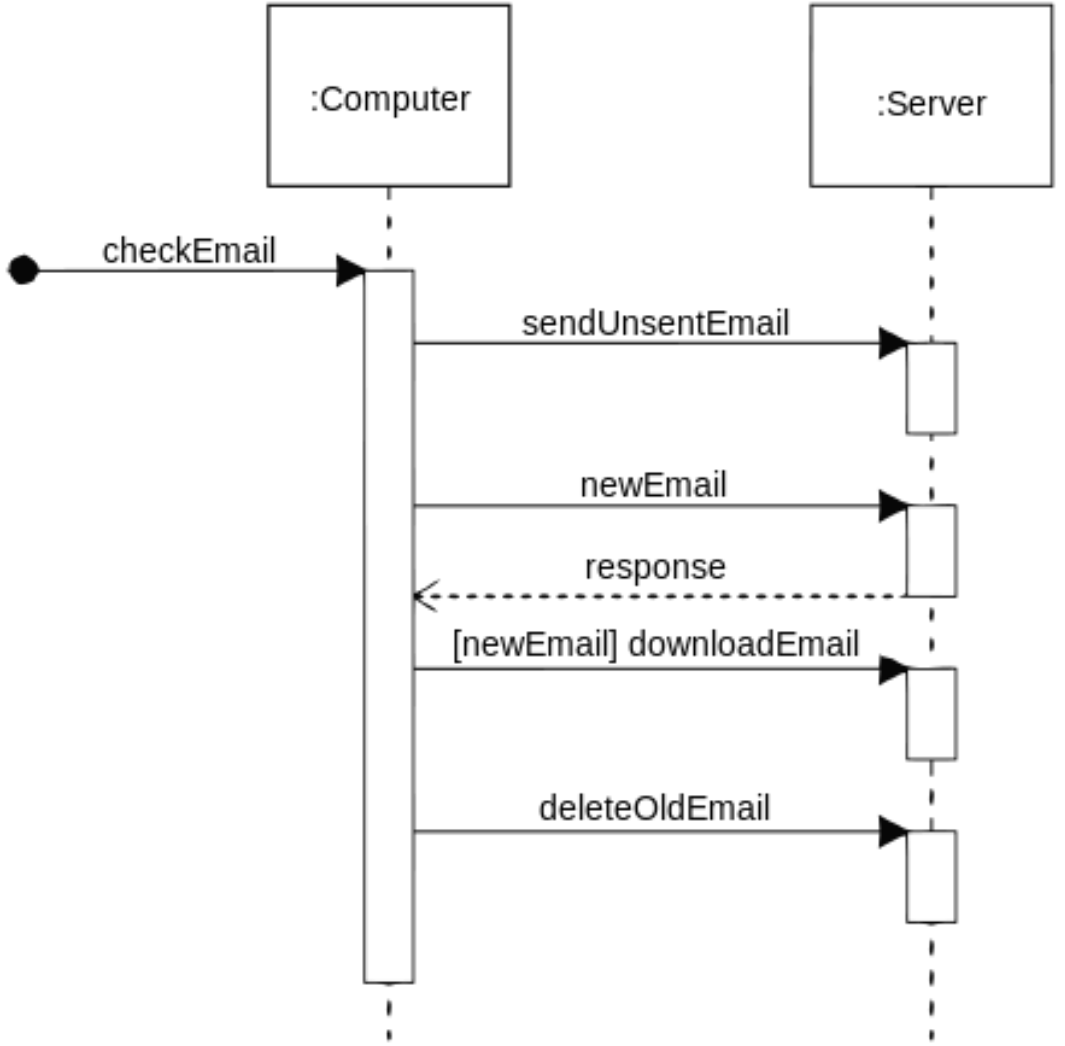
Use case diagrams

- Use case diagrams show relationships between users of a system and different use cases where the user is involved
- Example from Wikipedia:



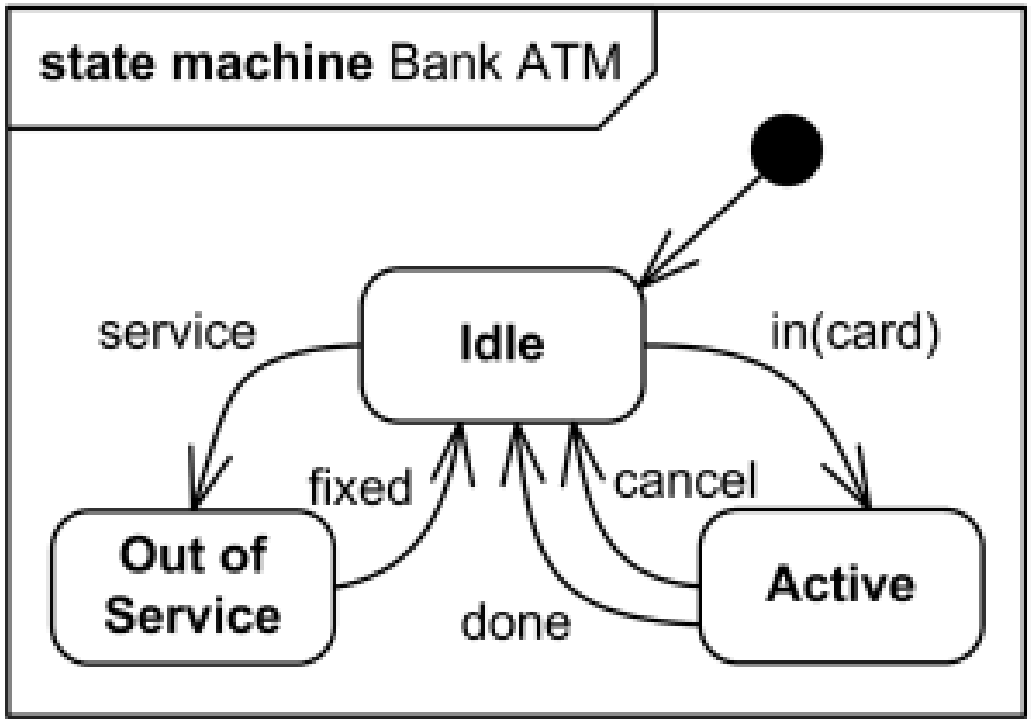
Sequence diagrams

- Sequence diagrams show system object interactions over time
- These messages are visualized as arrows
  - Solid arrow heads are synchronous messages
  - Open arrow heads are asynchronous messages
  - Dashed lines represent replies
- Example from Wikipedia:



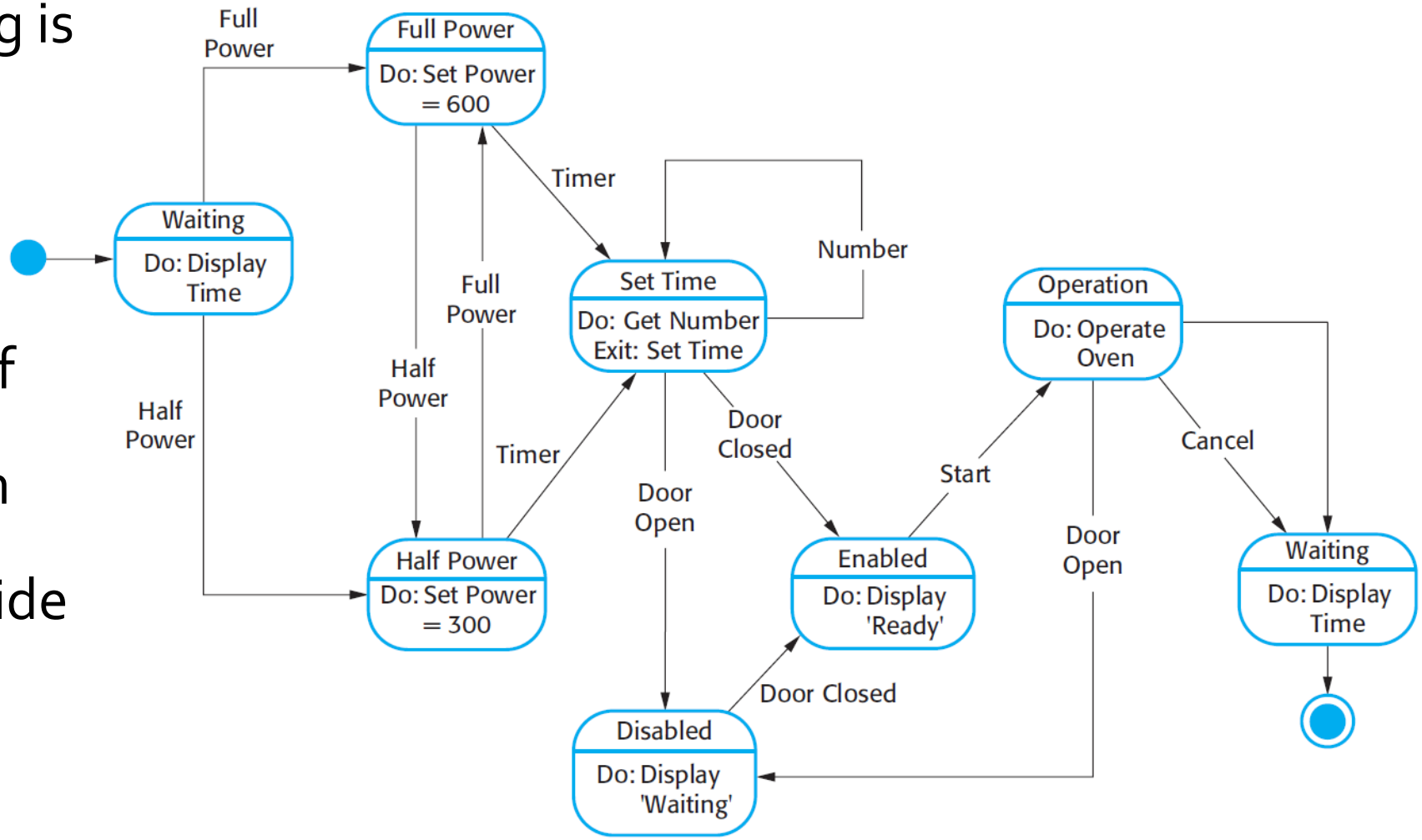
State diagrams

- State diagrams are the UML generalization of finite state automata from discrete math
- They describe a series of states that a system can be in and how transitions between those states happen
- Example from uml-diagrams.org:



Event-driven modeling

- Event-driven modeling is another kind of behavioral modeling that focuses on how a system responds to events rather than on processing a stream of data
- Here's a state diagram for a microwave oven based on various outside events



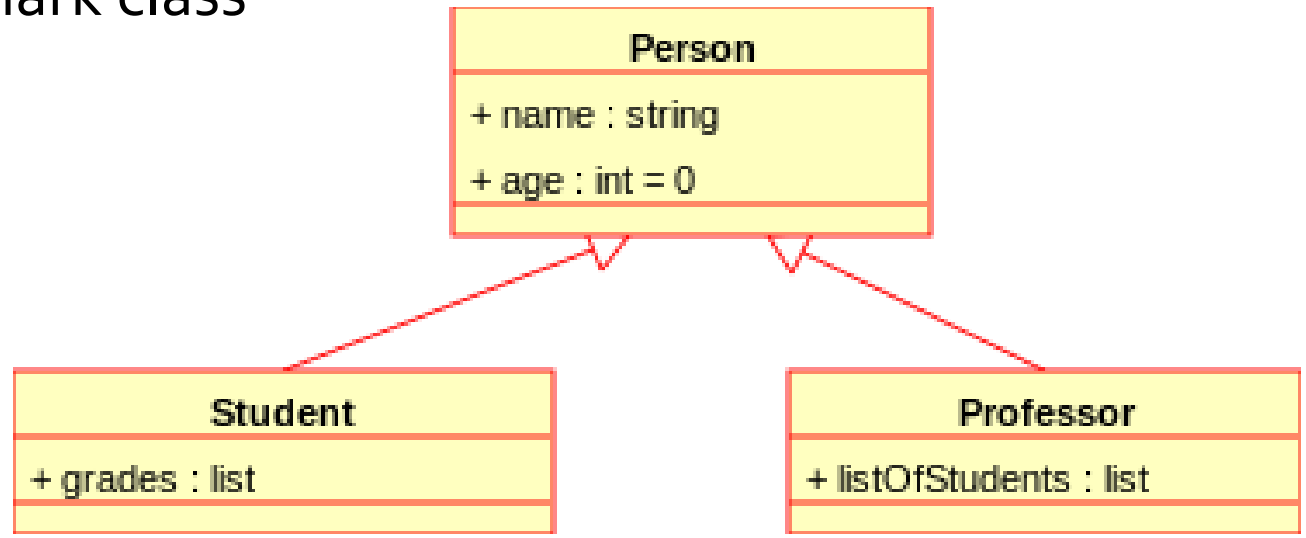
Class Diagrams

Structural models

- Structural models show how a system is organized in terms of its components and their relationships
- UML class diagrams are used for structural models, but they can be used in many different ways:
  - Relationships
  - Generalization
  - Aggregation

Class diagrams

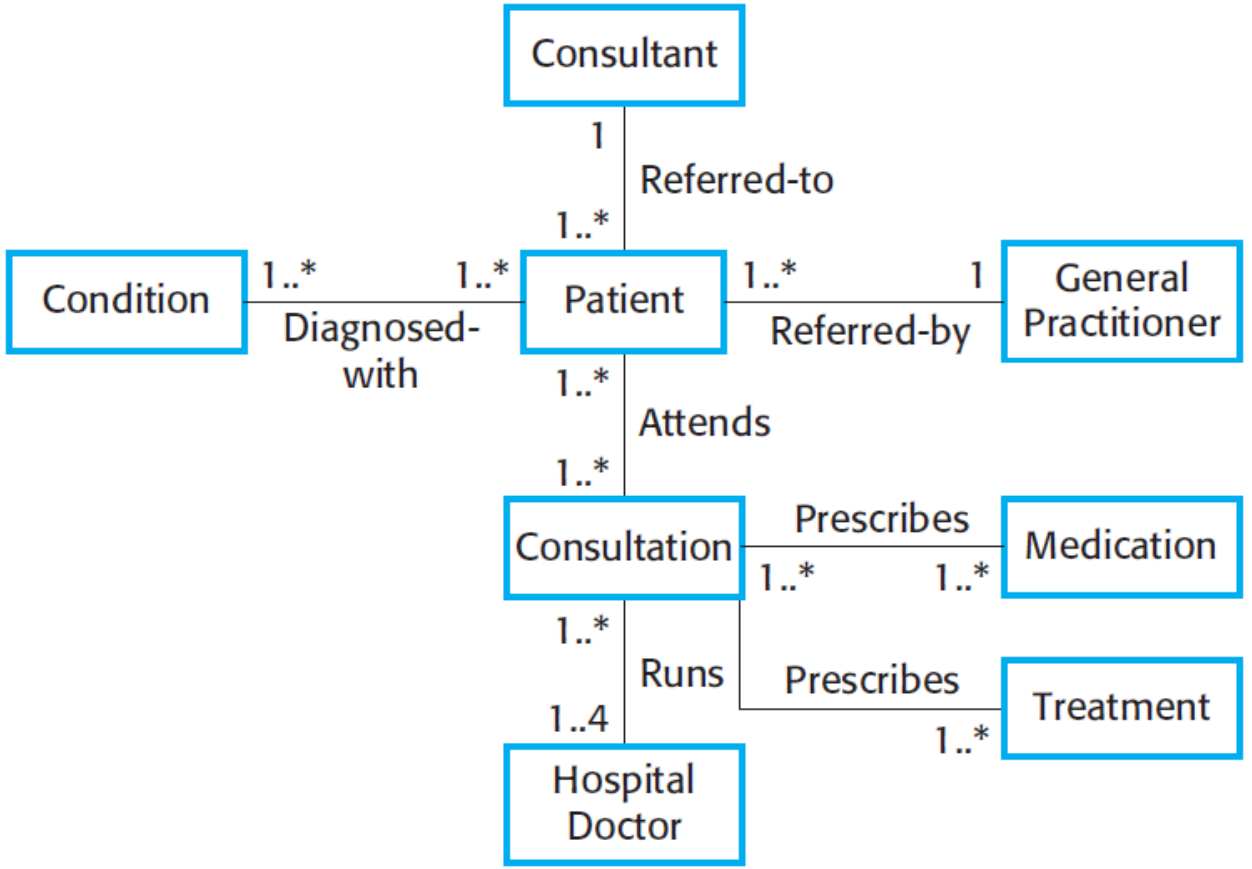
- Class diagrams show many kinds of relationships
- The **classes** being described often (but not always) map to classes in object-oriented languages
- The following symbols are used to mark class members:
  - **+** Public
  - **–** Private
  - **#** Protected
  - **/** Derived
  - **~** Package
  - ***** Random
- Example from Wikipedia:



Relationships

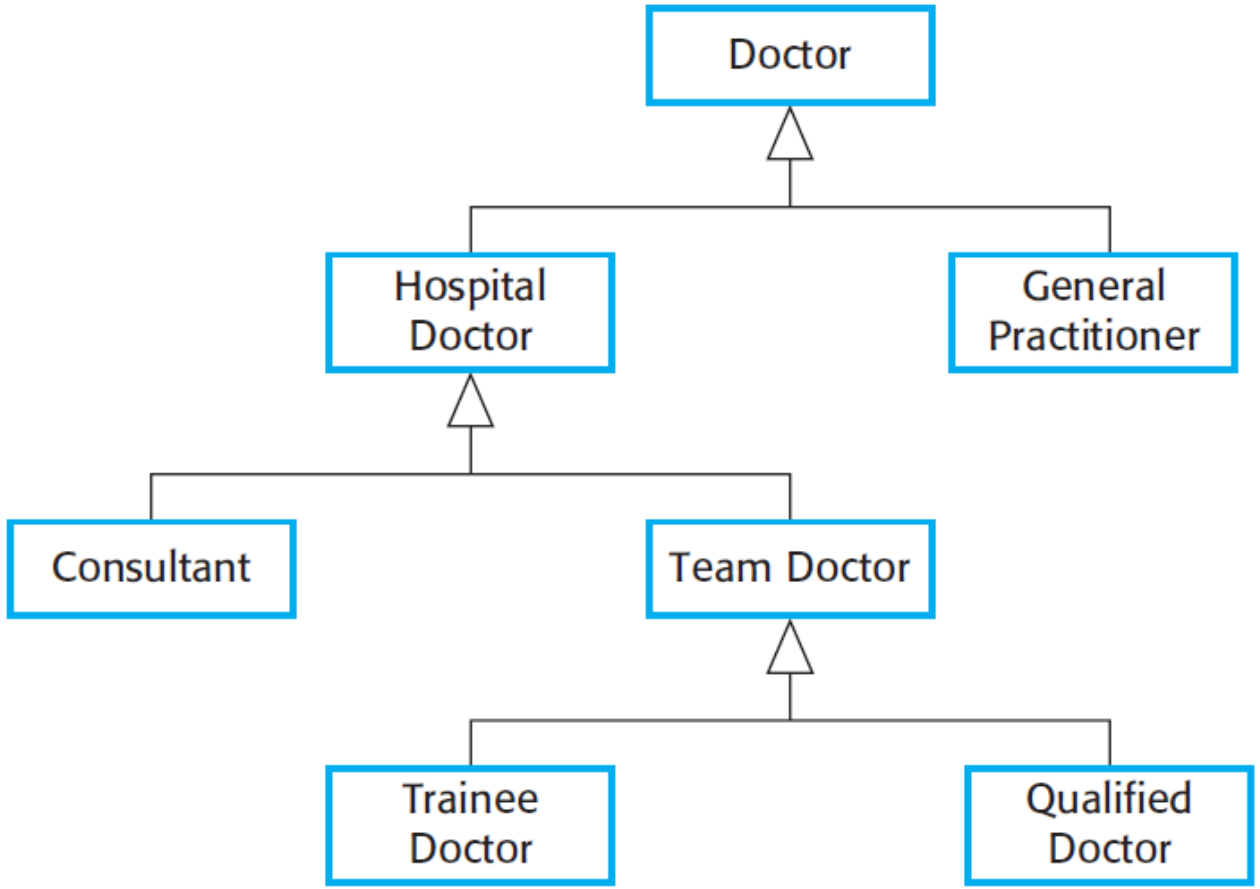
- Associations between classes can be drawn with a line in a class diagram

- Notations can be used to mark relationships as one to one, many to one, many to many, etc.
- These kinds of relationships are particularly important when designing a database



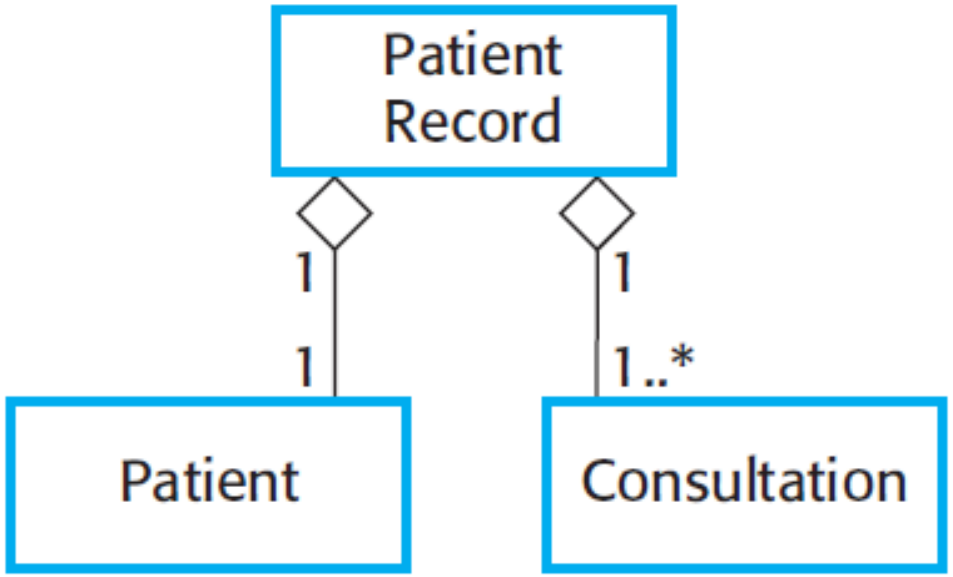
Generalization

- Classes can be listed with their attributes
- However, there are often classes that share attributes with each other
- Some classes are specialized versions of other classes, with more attributes and abilities
- This relationship between general classes and more specialized classes is handled in Java by the mechanic of **inheritance**



Aggregation

- Another way of using class diagrams is to show that some objects or classes are made up of smaller parts represented by other classes
- A diamond shape is used to mark a class that is the whole, and its parts are connected to the diamond



Upcoming

Next time…

- Read Chapter 2

CAREER JUMPSTART EVENT

Engineering & Computer Science

THURSDAY, SEPTEMBER 12TH FROM 4:45PM-7PM

Otterbein University @ The Point

Come and network with alumni and recruitment partners and learn how to be successful with your field.

OTTERBEIN
UNIVERSITY

SCAN the QR CODE to REGISTER

Reminders

- Read Chapter 2: Software Processes for Wednesday
- Keep working on your projects
  - SRS draft due Friday!