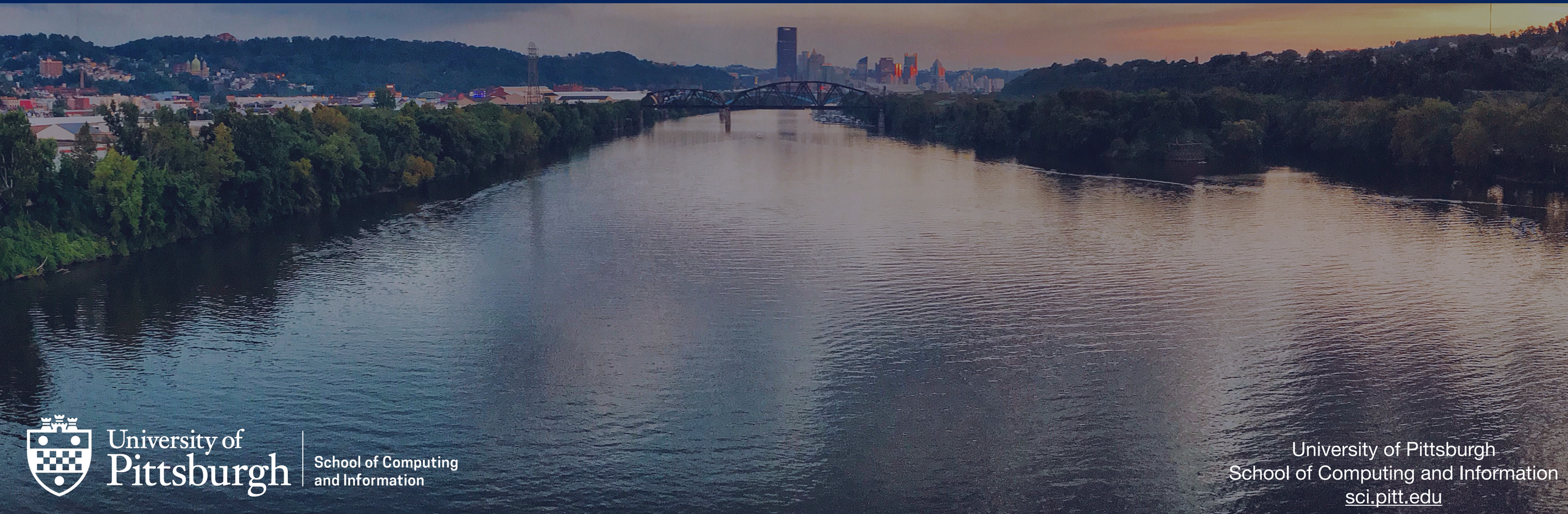
Fall 2025

L05 Model-based Engineering

CS 1530 Software Engineering

Nadine von Frankenberg



University of Pittsburgh
School of Computing and Information
sci.pitt.edu

Copyright

- These slides are intended for use by students in CS 1530 at the University of Pittsburgh only and no one else. They are offered free of charge and must not be sold or shared in any manner. Distribution to individuals other than registered students is strictly prohibited, as is their publication on the internet.
  - All materials presented in this course are protected by copyright and have been duplicated solely for the educational purposes of the university in accordance with the granted license. Selling, modifying, reproducing, or sharing any portion of this material with others is prohibited. If you receive these materials in electronic format, you are permitted to print them solely for personal study and research purposes.
  - Please be aware that failure to adhere to these guidelines could result in legal action for copyright infringement and/or disciplinary measures imposed by the university. Your compliance is greatly appreciated.
- Material from these notes is obtained from various sources, including, but not limited to, the following:
  - Bruegge, & Dutoit. Object–oriented software engineering. using UML, patterns, and Java. Pearson, 2009.
  - Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. Pearson, 1994.
  - Sommerville, Ian. "Software Engineering" Pearson. 2011.
  - http://scrum.org/

Learning goals

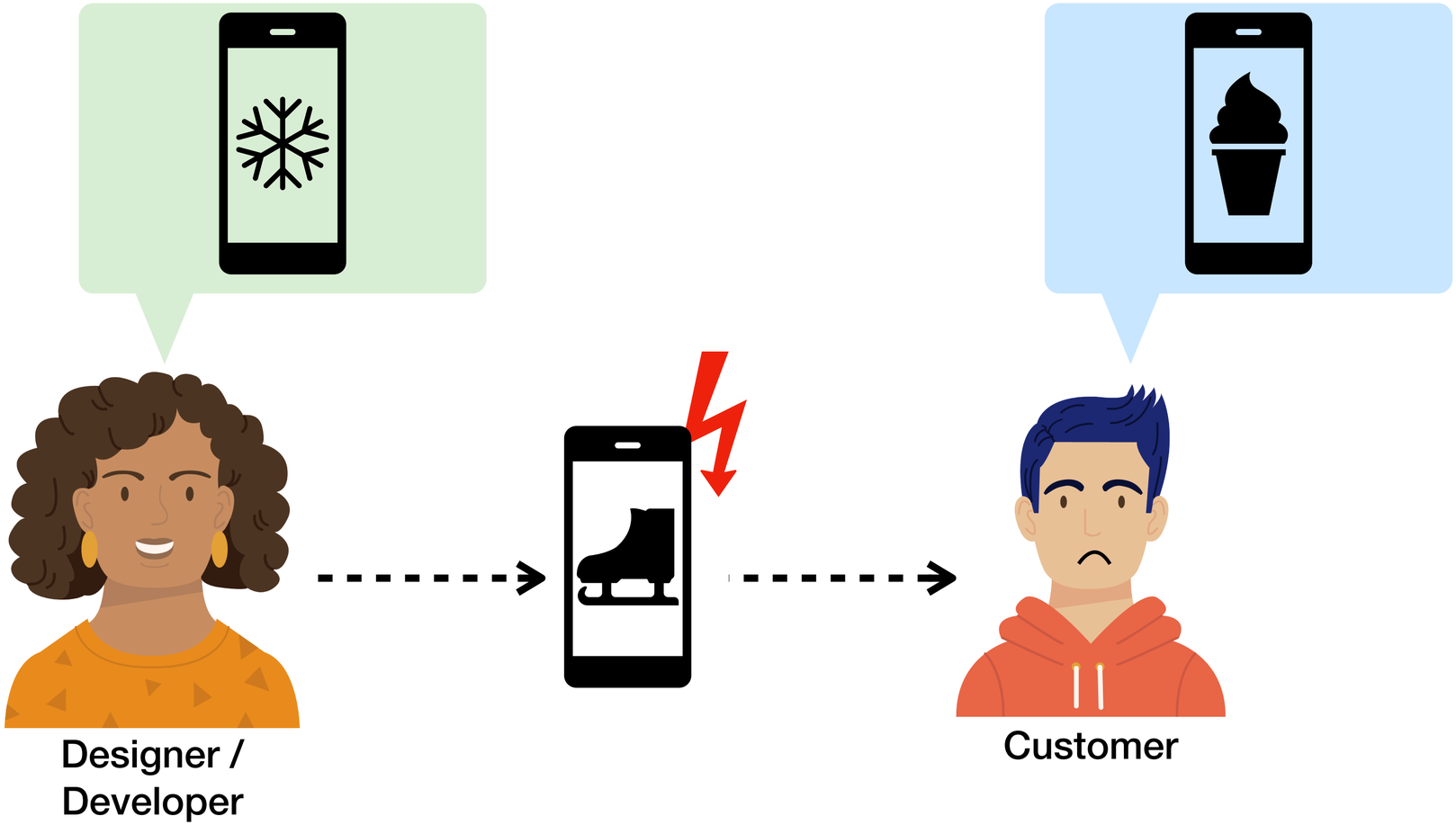
- You have a better understanding of the user vs enduser
- You understand how to extract relevant information from a problem statement
- You have an overview of requirements engineering

Today's roadmap

- The importance of abstraction
- Intro to model-based engineering
- Intro to requirements engineering

[Recap] Designing software systems

- Internal representation of external reality
- Typically



[Recap] Problems with system integrations

- Communication issues
- Wrong usage of available tools
- Interface miscommunication
- Antipatterns
- Physical impossibility
- Data inconsistencies
- Dependency conflicts
- Performance bottlenecks
- Unforeseen compatibility issues
- …

Holistic Approach:
To solve problems, we need a holistic view of the entire system

- Creative thinking
- Cross-disciplinary

Understanding the problem

- Engage with the customer about the problem scope and the requirements
- Identify specific components of the problem to ensure a comprehensive understanding
- **Divide-&-conquer!**
  - Break down complex problems into manageable, smaller components/pieces
  - Formulate a structured approach to tackle each component/piece individually
- **Problem statements** aid in recognizing distinct entities and elements
  - They enable the identification of unique objects for effective problem-solving

Basic assumption of software projects: project outcome cannot be produced in a single monolithic activity

"Unique objects" are distinct entities or components that have specific roles or attributes within a system. They typically contribute to the overall functionality of the system.

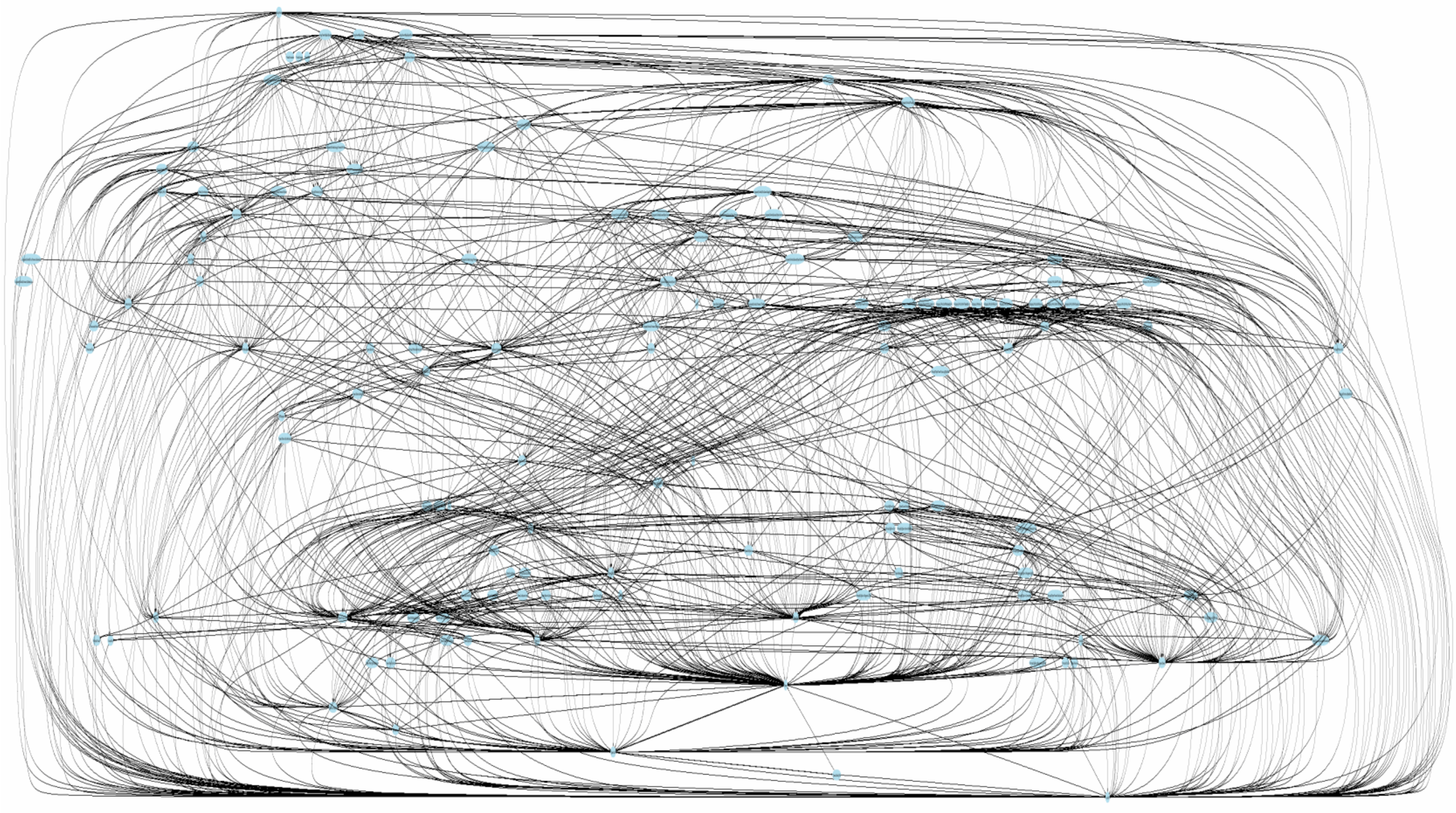
Problem Statement

- A problem statement is our **communication basis**
- It is a *concise description of the issue that the system aims to address*
- Usually, problem statements are documents that contain:
  - **Current situation**: What are the existing circumstances, if available
  - **Specification of functionality**: What capabilities are required?
  - **Delivery**: Where and how will the system be deployed?
  - **Customer expectations**: What are the customer's expectations and deliverables?
  - **Milestones**: Are there concrete delivery dates and project phases?
  - **Acceptance criteria**: What will we use to evaluate the system?

Problem Statement

We need a way so that people can get from one location to another using multiple means of transport, based on their available options and preferences.

Software systems are often very complex



Managing complexity in complex systems — Abstraction

- Complex systems are difficult or sometimes even impossible to understand
- "*The magical number **seven, plus or minus two***"

  Miller's law
  - Human Brain Constraint
    Our immediate memory span covers 7± 2 pieces of information at the same time
  - Widely applied in various fields to enhance information retention and usability
  - **[Example]** phone numbers (*not including the area code*)
- The **Chunking** technique
  - Group collection of objects to reduce complexity
  - **[Example]** instead of 5-6-5-1-9-7-4 chunk into 565-1974

Abstraction is key

- Allows us to hide unessential details and focus on essential aspects
- Abstraction is a thought process (activity) where ideas are distanced from objects
  - E.g., We break a system into **layers** (UI, logic, storage) where each layer abstracts away details not necessary for the other layers to function
- Abstraction is the result (entity) of a thought process
- Abstractions can be expressed with a **model**

Today's roadmap

- The importance of abstraction
- Intro to model-based engineering
- Intro to requirements engineering

Models are abstractions of systems

University of Pittsburgh

- An **existing system**
  - **[Example]** Integrate a feature or troubleshooting for Canvas or TopHat
  - Often used in reverse engineering of a system that lacks clear communication
- A **system to be built**
  - **[Example]** Create user workflows/recommendation logic for a restaurant picker app
  - Most common: UML diagrams, wireframes, and state models
- A **system that no longer exists**
  - **[Example]** Modeling legacy systems (Netscape web browser, MacDraw, Facebook Paper, …) could help inform if a modern replacement / support documentation
  - Also useful for data migration, compliance, research

Typical models in software engineering

University of Pittsburgh

- **Functional model**: What are the functions of the system? (**use cases**)
- **Object model**: What is the structure of the system? (**entities**)
- **Dynamic model:** How does the system react to external events? (**activities**)

- **System model**: object model + functional model + dynamic model

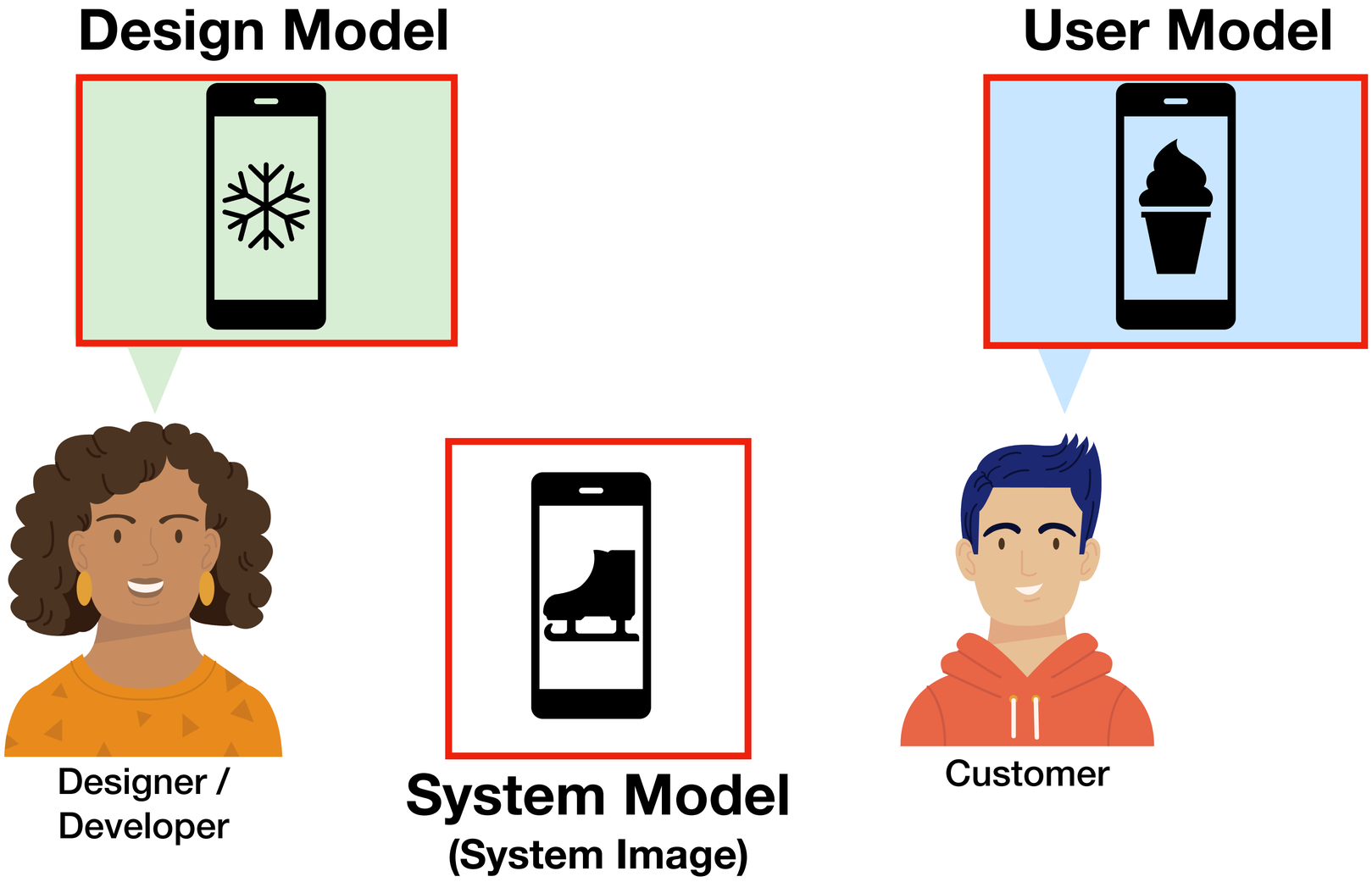
What is **model-based engineering**?

University of Pittsburgh

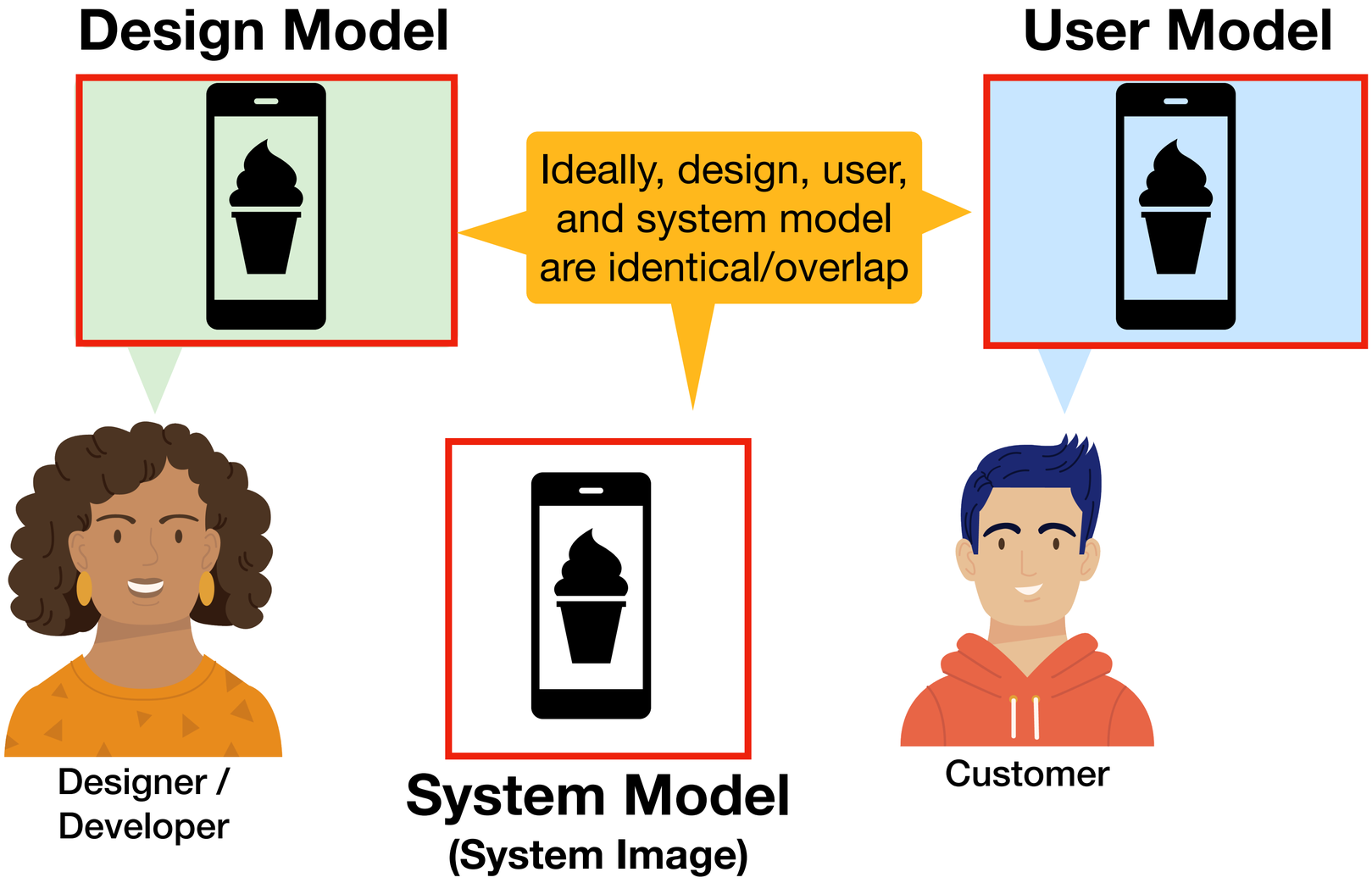
- Models are **abstract representations** of system behavior and structure
- **Visual and standardized way** of presenting complex systems
- Used to **design**, **analyze**, and **document systems**
- Supports **iterative development**
- Can be used as tool for **communication** between stakeholders
  - Also used in other fields, e.g., mechanical engineering

Models can help guide the development of systems.
E.g., the general code structure (important classes) can be derived from class diagrams

Problem with models



Problem with models



Mental models

What users believe they know about a user interface impacts how they use it. Mismatched mental models are common, especially with designs that try something new.

- **User model** — *what the user thinks the system does*
  - The user's expectations of the system
  - Reflects how users perceive and understand the system
- **Design model** — *what the developers intended the system to do*
  - Created by designers and developers based on the system's functionality
  - Typically before coding begins
- **System image** — *what the system actually does/conveys through UI/behavior*

[Example] ATM - User Model Assumptions

- Users expect an ATM to dispense cash quickly and securely after they input their PIN and select the amount
- Assumptions
  - The ATM will process the user's request without delays
  - User can cancel a transaction at any point if needed
  - User's card will always be returned after use

https://web.mit.edu/2.744/studentSubmissions/humanUseAnalysis/jsaadi/images/pin pad.jpg

https://fs.hubspotusercontent00.net/hubfs/8405653/atm-machine-and-money-withdrawing-dollar-banknote.jpg

[Example] ATM - Design Model (*simplified)

- Users will follow on-screen instructions/prompts carefully step-by-step
- The system includes safeguards
  (e.g., locks the card if a user inputs the wrong PIN multiple times)
- Cash dispensing happens only after confirming sufficient account balance

[Example] ATM - System Image (*simplified)

- A series of prompts for language selection, PIN entry, and transaction type
- A delay between PIN entry and the display of account options due to backend processing
- A button for canceling a transaction

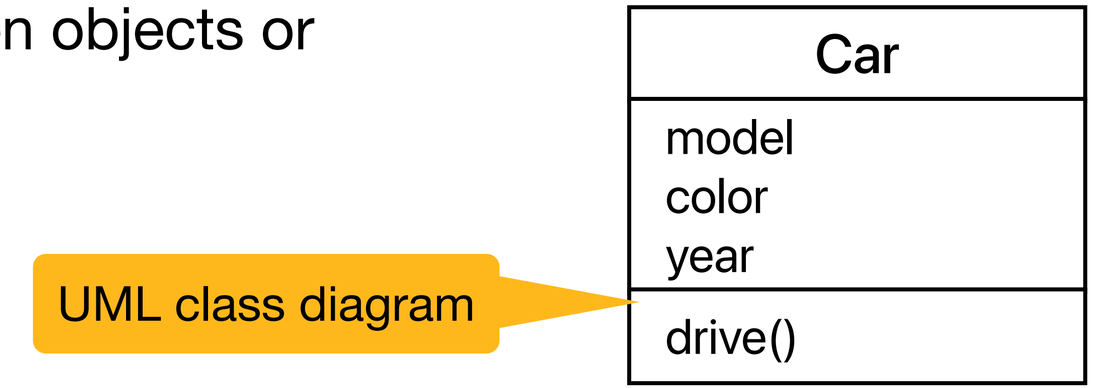
https://languagelog.ldc.upenn.edu/~bgzimmer/atm.jpg

Problems with models

- **Problem**: The more stakeholders involved, the more "mental models" exist
  - Requires extensive communication
  - Prone to misunderstandings
- **Solution**: Use a **common language** to **abstract and describe problems**
  - Ensures a common understanding
  - Refactor models as needed

Unified Modeling Language

- Created in the late 1990s by a consortium of software companies
- Became an official **standard** in 2005 by the Object Management Group (OMG)
- UML diagrams provide a **visual representation** of an aspect of a system
- UML includes a variety of diagram types for modeling different aspects of software systems
  - **Structural** diagrams → static structure
  - **Behavioral** diagrams → dynamic behavior
  - **Interaction** diagrams → interactions between objects or components



Why UML?

- Reduces complexity by focusing on abstractions
- Allows for a better organization of code
- Provides a communication basis
  - Common vocabulary
- Analysis and design
  - Enables to specify a future system without focusing on implementation details

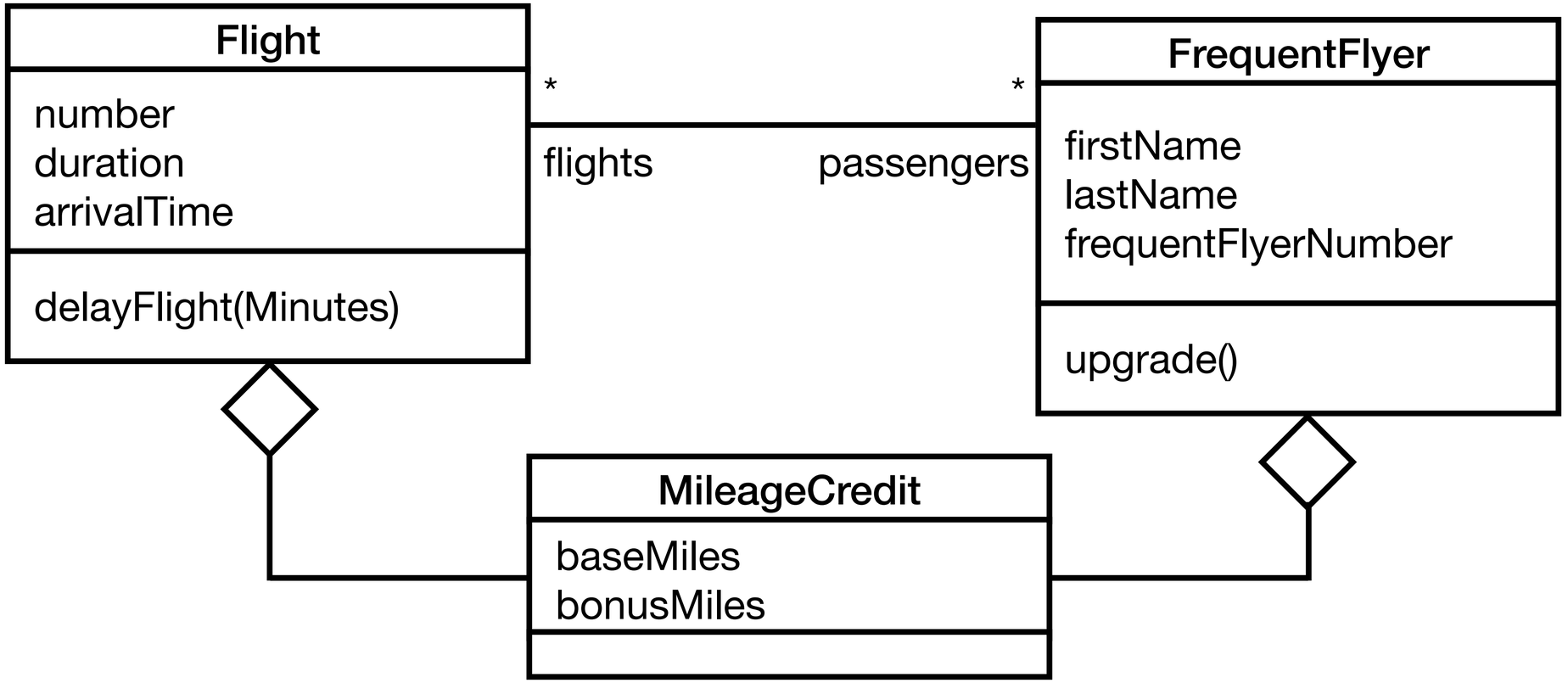
Purposes of UML diagram types

- Use case diagram: shows functionality
- Class diagram: shows class/object structure
- Activity diagram: shows dynamic behavior (activities & objects)
- State chart: shows dynamic behavior (states of an object)
- Communication diagram: shows interaction (messages between objects)
- Sequence diagram: shows interaction (method invocation & lifespan)
- Deployment diagram: shows system architecture
- …

UML in practice

- Documentation
  - Many companies use UML to document their systems
    - Ensures global understanding
  - Many open-source projects include UML models in their documentation
- Ideation & design thinking
  - Structured approach in defining ideas
- Programming
  - Allows to share ideas
  - Helps in defining complex algorithms
  - Provides starting point for writing and organizing code
- Enhances abstract thinking & builds problem-solving skills

[Example] FrequentFlyer — UML class diagram



This is an example of a UML class diagram of a frequent flyer system.
We will talk more about the individual UML elements soon!

Two abstraction domains

University of Pittsburgh

- **Application domain** (= problem space)
  - Focus is on understanding and anlyzing the **problem**
  - Environment where the system is operating

- **Solution domain** (= solution space)
  - Focus is on **design** & **implementation**
  - Technologies used to build the system

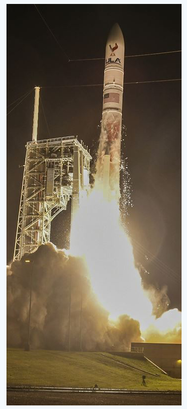
[Example] Application vs. solution domain

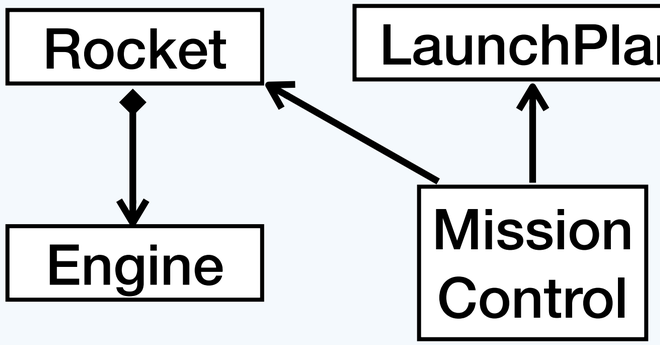
University of Pittsburgh

Reality

Model

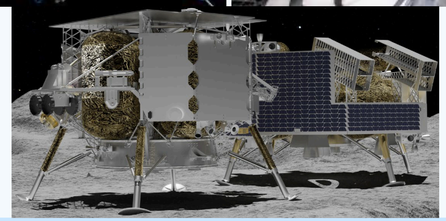
Application Domain

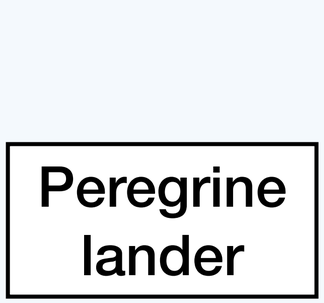


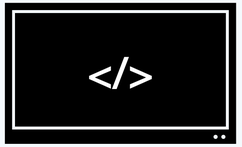


Analysis

Solution Domain







Design & Implementation

Overview of Model-based Software Engineering

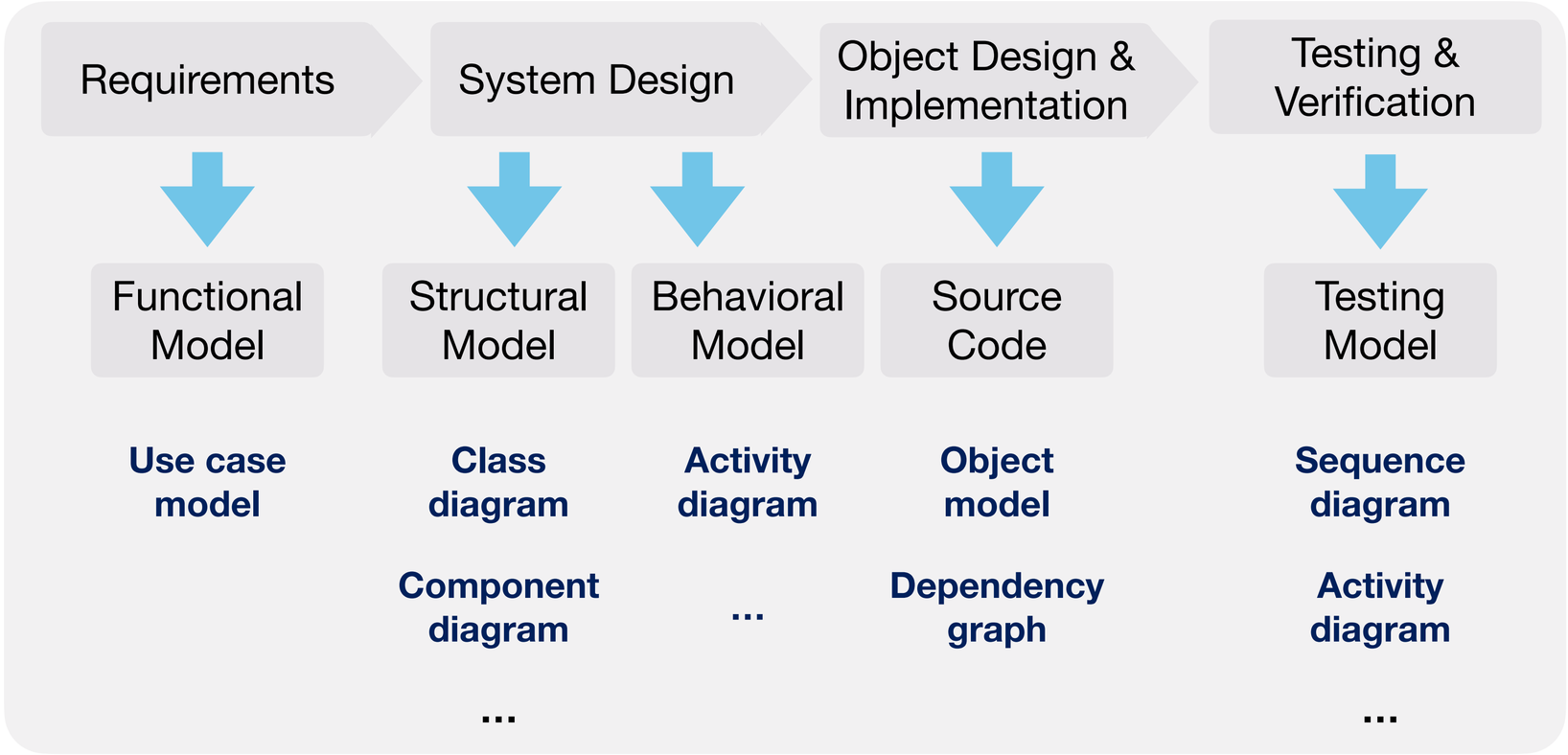
University of Pittsburgh

Problem

Commuter App

Problem Statement

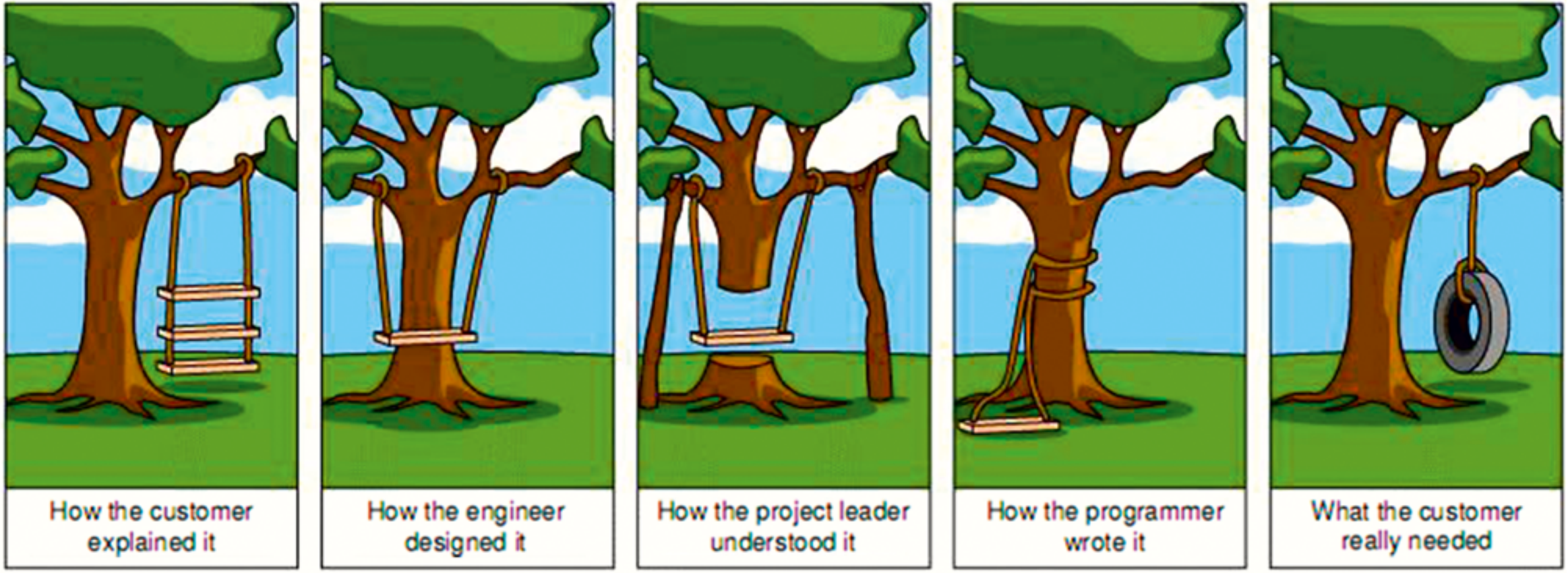
We need a way so that people can get from one location to another using multiple means of transport, based on their available options and preferences.



Enduser

Today's roadmap

- The importance of abstraction
- Intro to model-based engineering
- Intro to requirements engineering


How the customer explained it
How the engineer designed it
How the project leader understood it
How the programmer wrote it
What the customer really needed

System requirements

- Requirements describe the purpose of the system
- They specify the **functionality** and **constraints** of a system
- **Requirements elicitation** describes the process of identifying requirements
- The elicited requirements are then analyzed and a system model is created

Note: Requirements are important but not set in stone!
Often, during system design or implementation,
new requirements arise or existing ones need to be adapted

Requirements Engineering

- Combination of requirements elicitation and requirements analysis
- An activity that defines the requirements of the system under construction
- **Requirements elicitation** defines the system from the **view of the** (end)**user**
  → Requirements specification
  → Uses natural language
- **Requirements analysis** defines the system from the **view of a developer**
  → Technical specification
  → Uses (semi-) formal language, e.g., UML

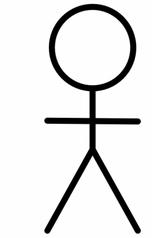
Requirements Engineering

= Process of eliciting requirements and analyzing them

- Activities
  - Identify actors (different users of the system)
  - Identify scenarios (in natural language describe the functionality of the future system)
  - Identify functional requirements
  - Identify non-functional requirements
  - Derive use cases (generalize functionality, describe their behavior, and identify the relationships among use cases)
  - Derive entities, characteristics, and behavior
  - …

Identifying actors

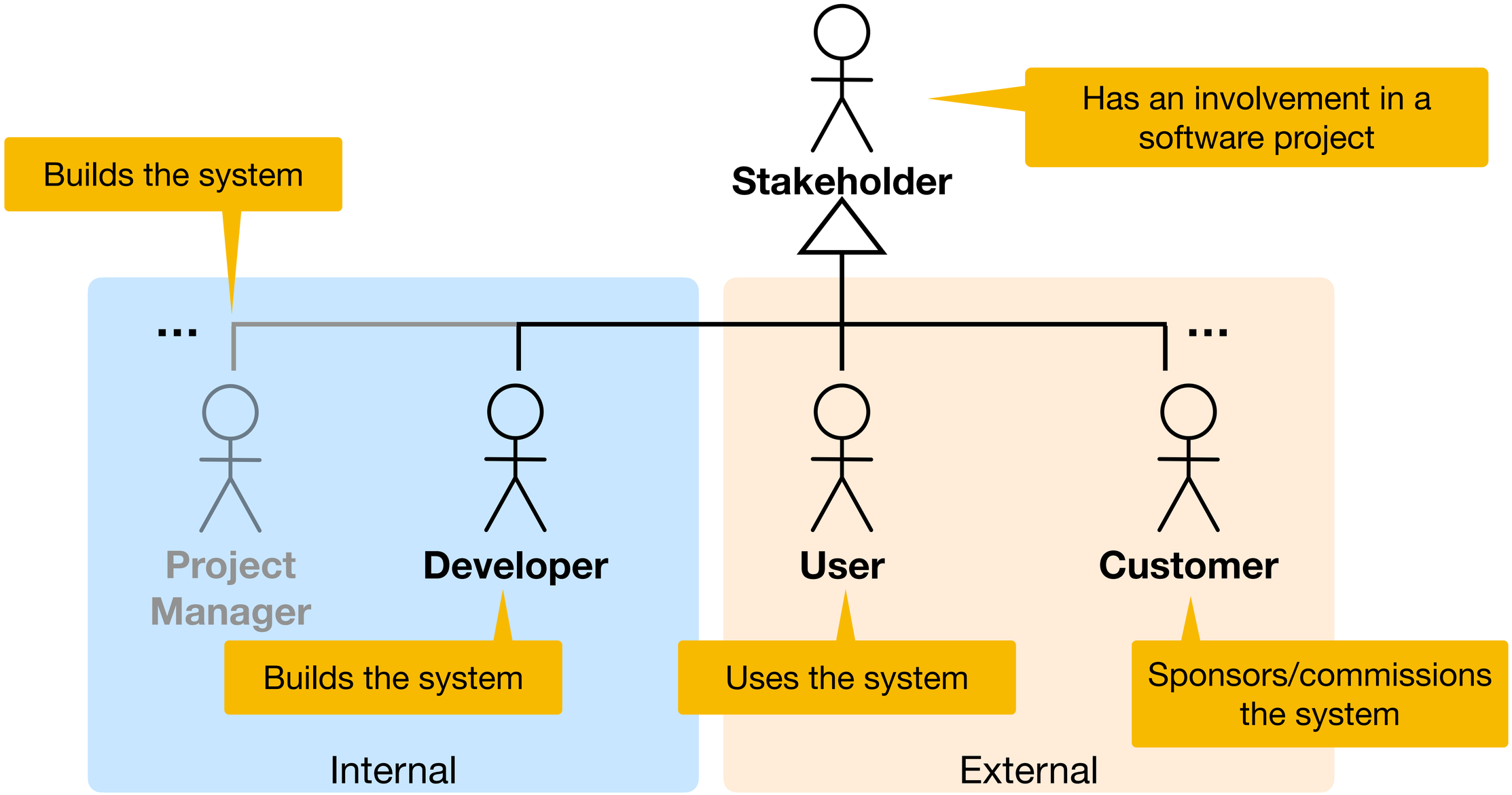
- Actors represent the different users of a system
  - Note that a user is not necessarily just the enduser!
  - Users can also include: system analyst, database admin, another system, …
- In UML, actors are represented as a **stick figure**



Actor

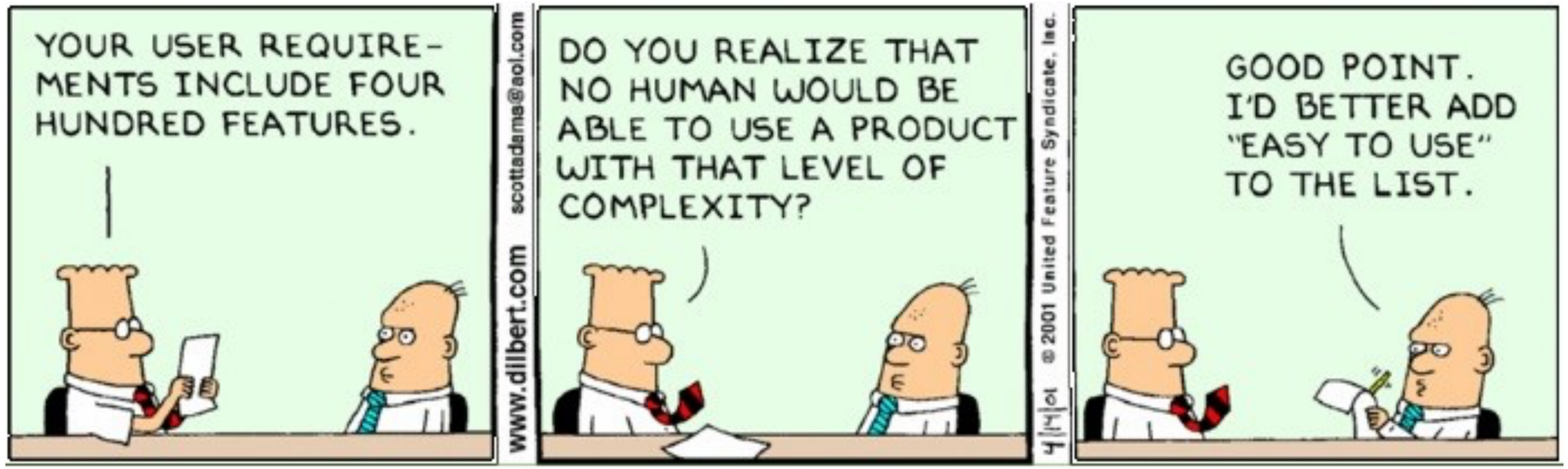
Should be a concrete user, e.g., "Student", "PizzaBaker", "Developer", …

Stakeholders in a software project



Take-Away: Overview of requirements elicitation

- User-Centric Approach: Prioritize understanding the needs and perspectives of end-users and stakeholders
- Effective Communication: Establish clear and open channels of communication to gather accurate requirements
- Iterative Process: Requirements elicitation is an ongoing, iterative process, adapting to evolving project needs
- Documentation: Thoroughly document gathered requirements to ensure clarity and alignment throughout the project lifecycle

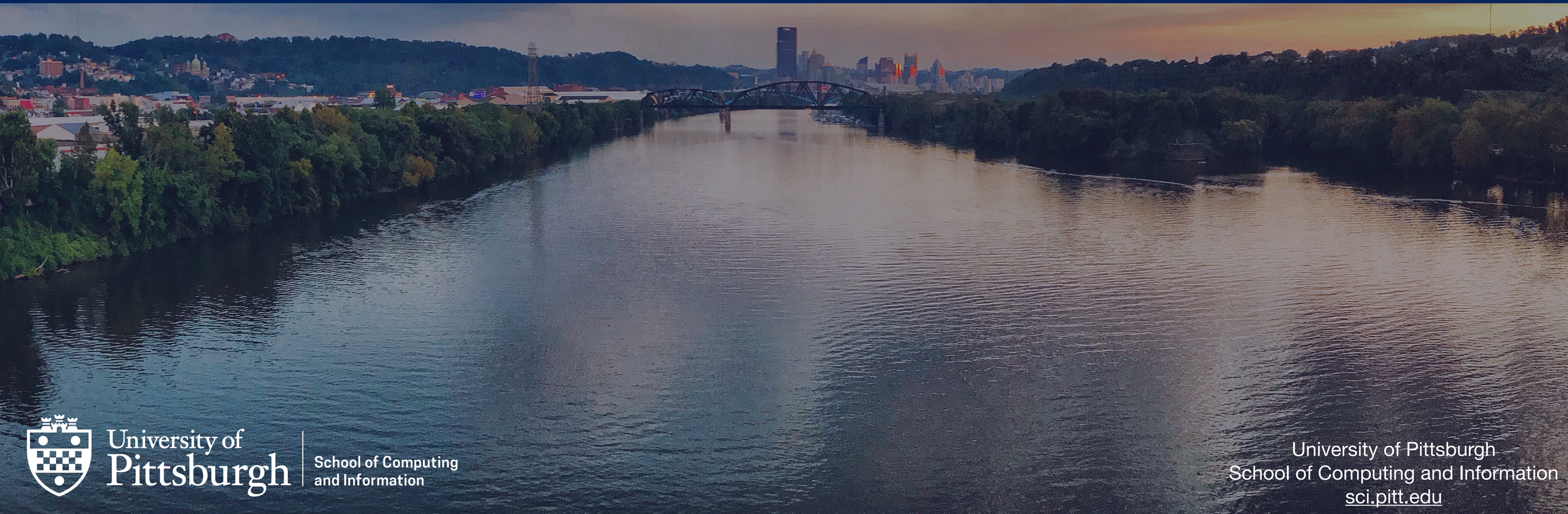


Fall 2025

L05 Model-based Engineering

CS 1530 Software Engineering

Nadine von Frankenberg



University of Pittsburgh
School of Computing and Information
sci.pitt.edu