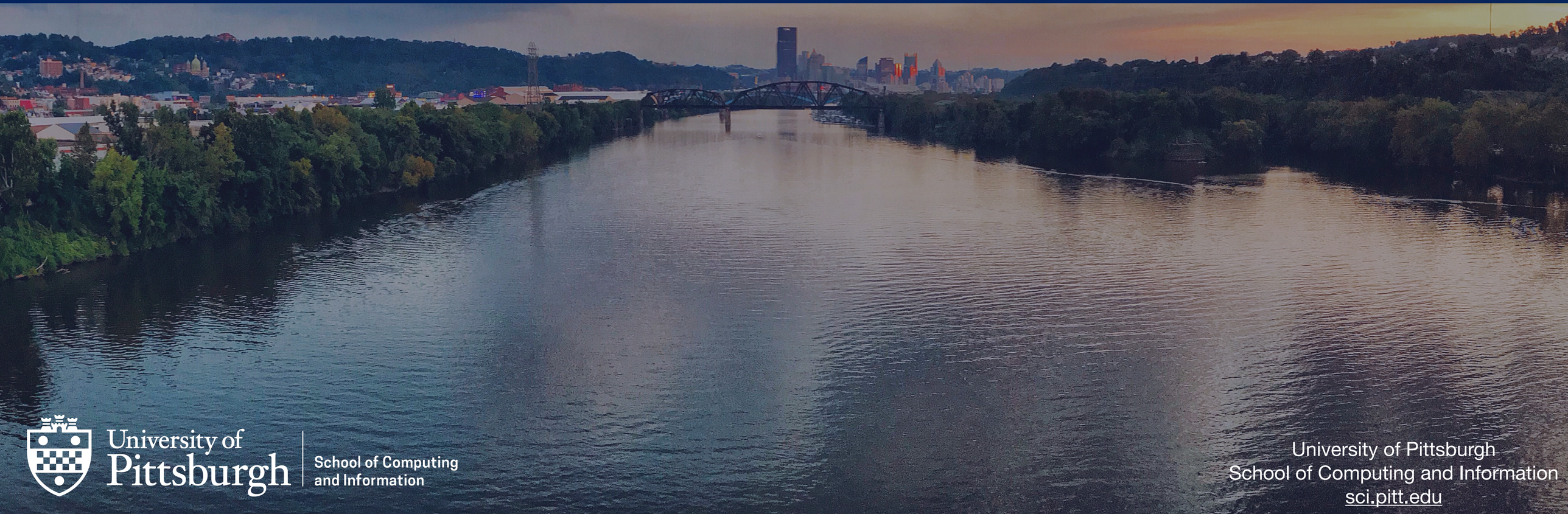
Fall 2025

L06 Requirements Elicitation I

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 user, enduser, and customer
- You can extract functional requirements from a problem statement

Today's roadmap

- Intro to requirements engineering
- Functional requirements

[Example] Who is the user, enduser, and customer?

University of Pittsburgh

Company A builds a software product for Company B.
Kim purchases the software product from Company B.

Company X develops an ordering platform for Restaurant Chain Y.
Restaurant managers use an admin dashboard to update menus & track orders.
Customers place orders through the app and get their food delivered.

Company M creates an online banking backend for Bank N.
Bank tellers and support staff use the system to manage accounts.
Bank customers use the mobile app or website to transfer money and check balances.

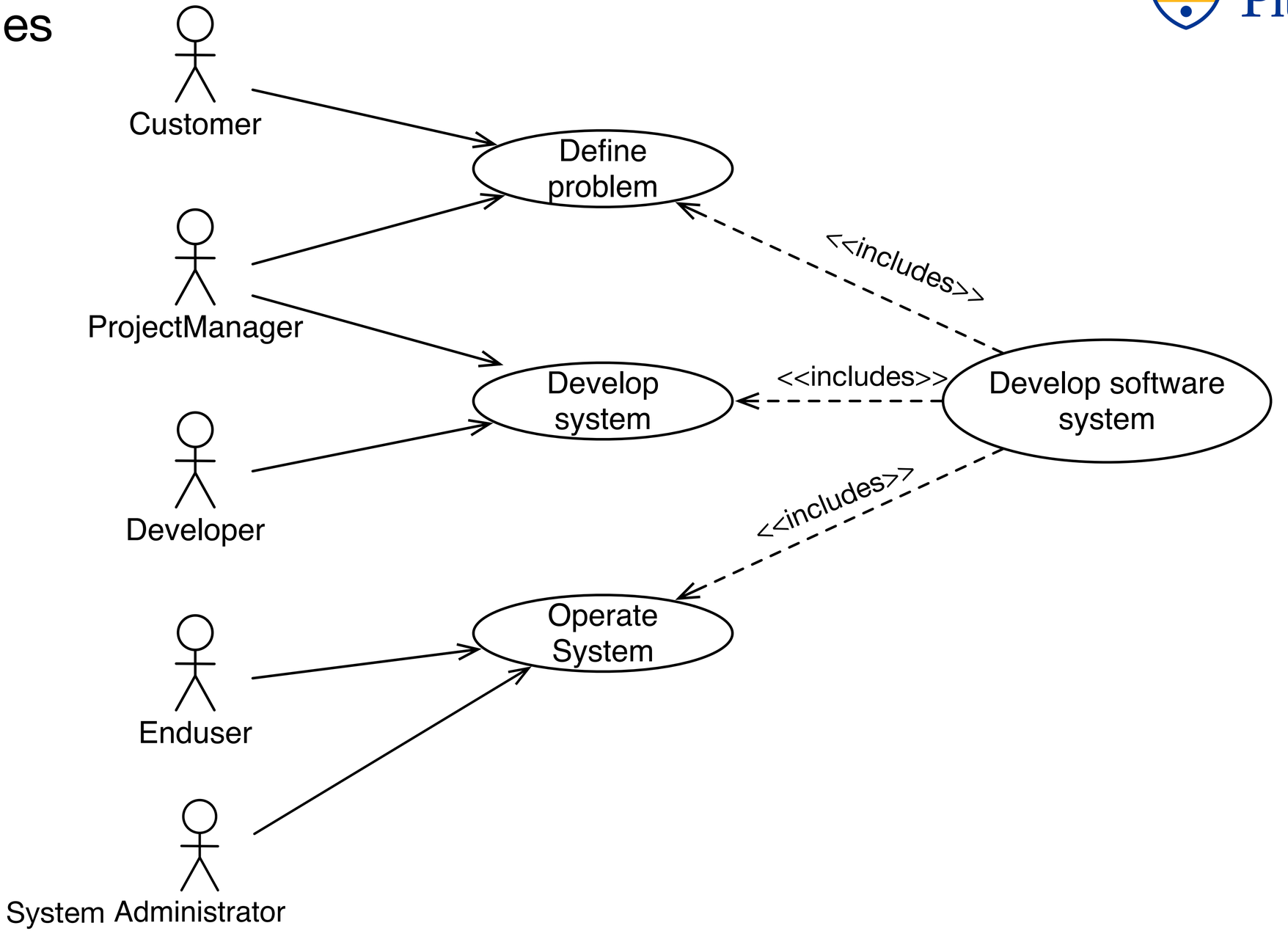
User vs enduser

- **User** is anyone who interacts with the system
  - Includes maintainers or supporters of a system
  - Can be an external system
  - **[Example]** system operations / administrators, technical personnel
- **Enduser** is the person who is intended to use the software for its primary function
  - Does not need advanced technical knowledge of the system
  - **[Example]** person who downloads an app from the App Store/Google Play Store

The users of a software system should be defined
at the start of a project!

© 2025 Nadine von Frankenberg

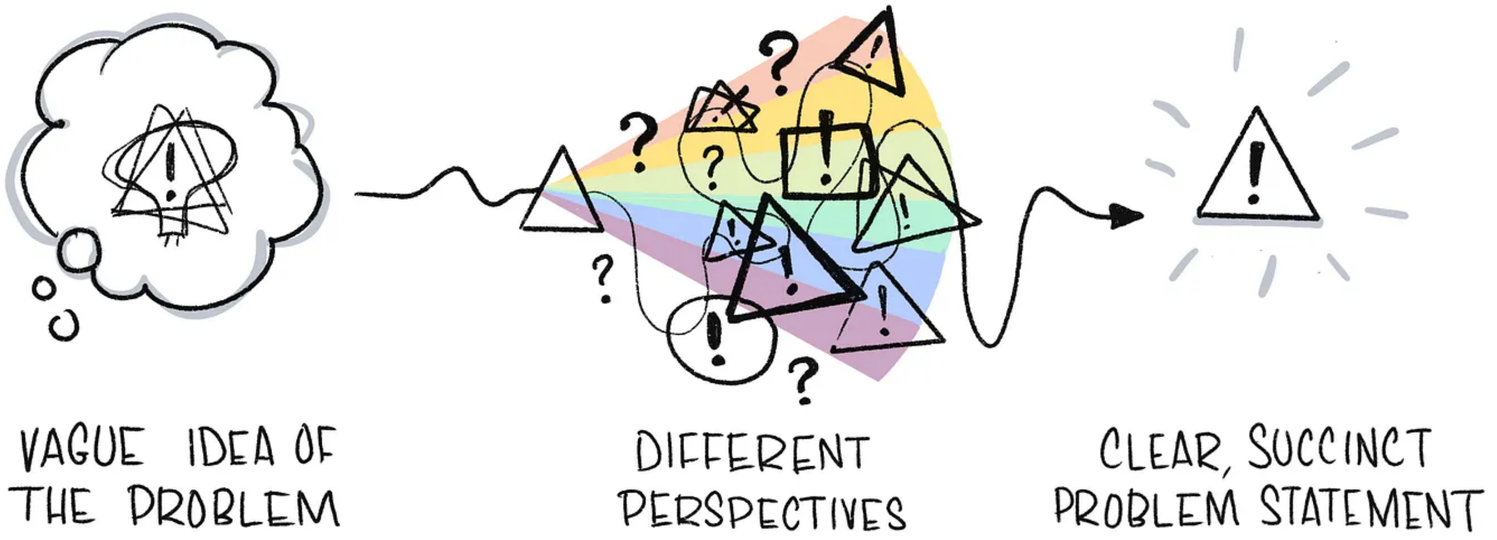
SDLC
Actors & Activities



Problem definition

- Typically documented as an informal problem statement
- Describes the purpose of the system
  - Not necessarily something that is wrong
  - Could describe the gap between an existing and desired state
  - Could be an idea
- **The problem is not the solution!**

Essential for building any system!

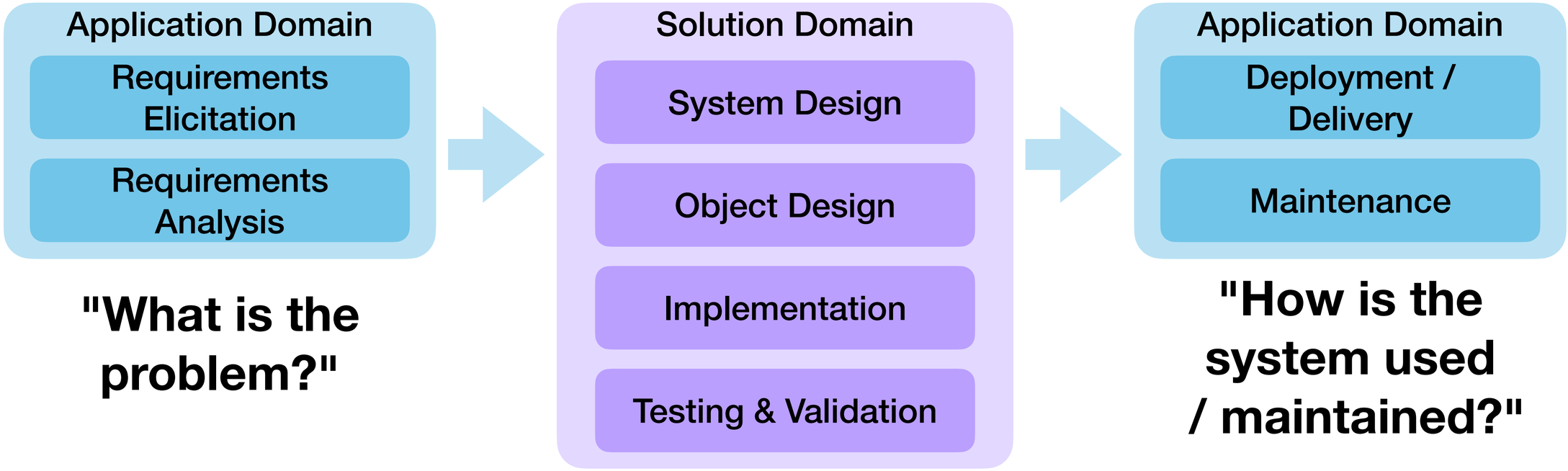


https://blog.prototypr.io/the-problem-with-problems-and-a-better-way-to-frame-them-visually-b3bb71f30baf

Why is it important to separate the problem from the (concrete) solution?

- **Clarity and understanding** — Avoid premature solution
  - Ensure well-informed solutions
- **Flexibility** — Encourage creativity and innovation
  - Explore various solutions
- **Communication** — Foster better collaboration
  - Enable a common understanding of the problem

Application vs Solution Domain



© 2025 Nadine von Frankenberg

[Example] Application vs Solution Domain

Phase	Example
Problem Definition	Students complain they can’t find parking spots near class on time
Requirements Elicitation	Interview students: Need real-time lot status, navigation to nearest open spot
Requirements Analysis	Create a use-case diagram for ‘Find Spot’ & model state machine for lot occupancy
Solution Domain – Design	Decide: mobile app + backend API with sensor data
Implementation	Build app using Flutter, connect to database
Testing/Validation	Check that parking spot data updates every 10 seconds
Deployment/ Maintenance	Release to App Store, monitor usage, fix bugs

© 2025 Nadine von Frankenberg

Different domains require different approaches

Greenfield Engineering

- Developing a system from scratch; no prior system exists
- Requirements: Extracted from stakeholders and (end)user input
- Initiation: New market needs
- Starting Point: Clear problem statement based on user needs

Different domains require different approaches

Re-engineering — redesign or re-implementation of an existing system

- Requirements: Driven by new technology
- Initiation: Responds to technological advancements
- Starting point: Problem statement rooted in technological evolution

Different domains require different approaches

Interface Engineering — adapting an existing system to a new environment

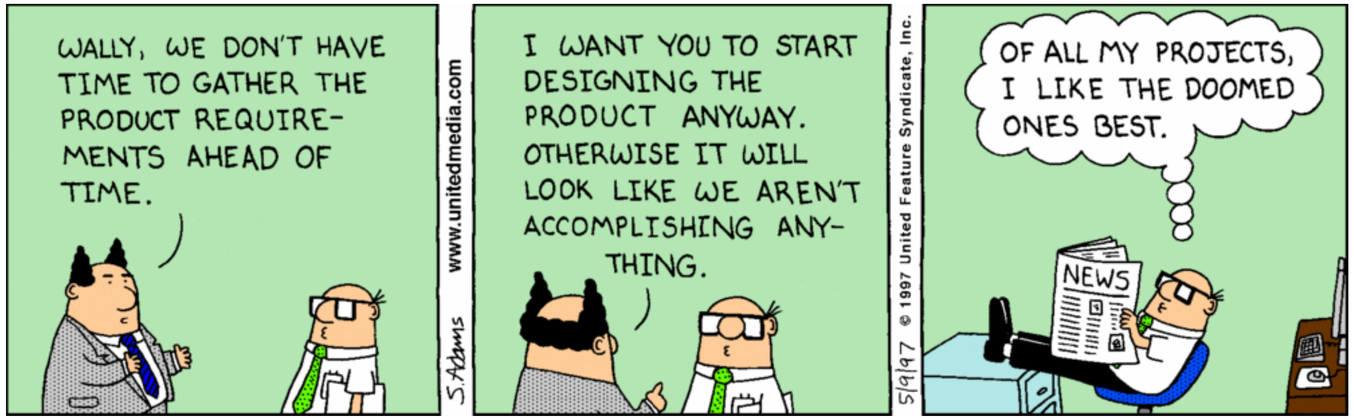
- Requirements: Prompted by technology or new market needs
- Initiation: Responds to technology shifts or emerging market demands
- Starting Point: Problem statement reflecting technological or market changes

Today's roadmap

- Intro to requirements engineering
- Functional requirements

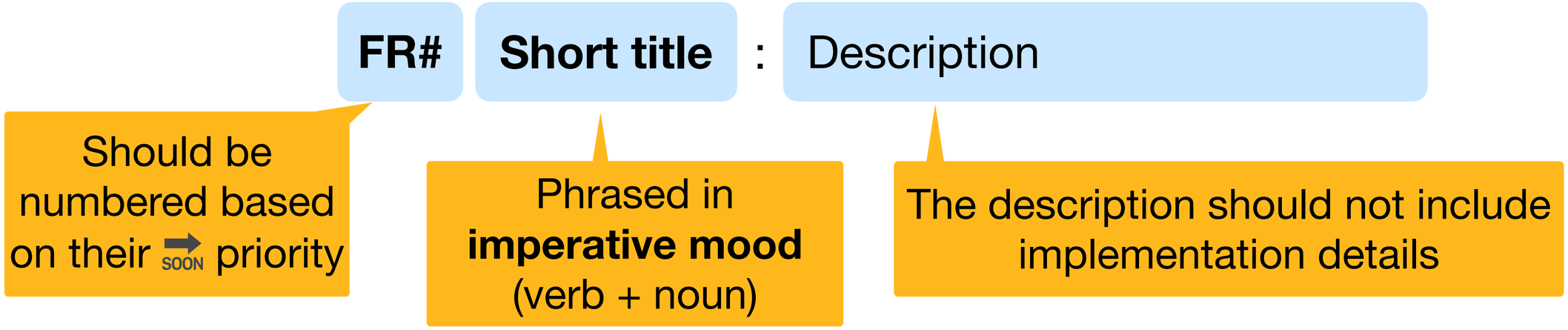
System Requirements

- Describe the functions to satisfy the stakeholder needs and requirements
- Basis for designing a software system
- Describe how the user requirements should be provided by the system
- Expressed in textual statements
- Types of requirements:
  - **Functional** (Focus is on the functionality)
  - **Quality** / **Non-functional** (Focus is on quality attributes)
  - **Constraints** (Focus is on limitations)



Functional Requirements

- Describe the essential functionality of the system
- Define one requirement per function (feature)
- They should be phrased in a general manner



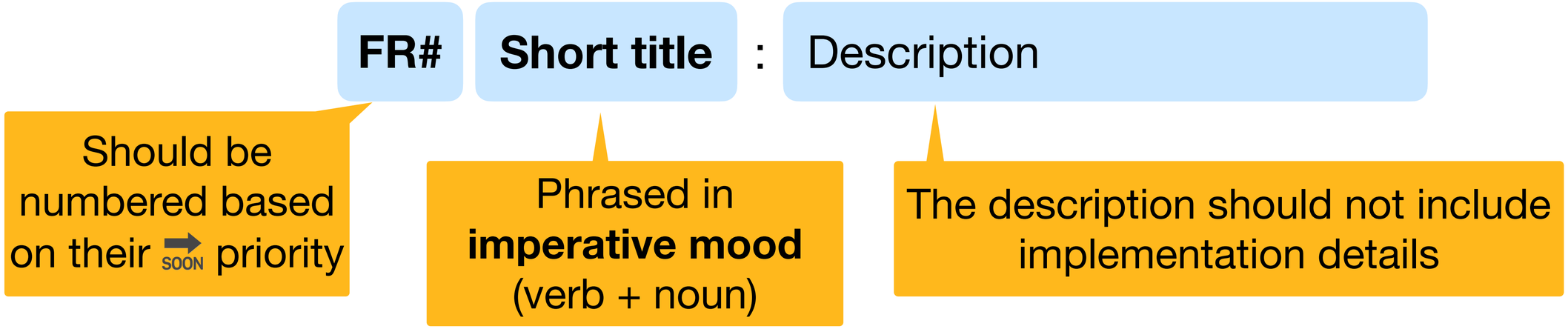
[Example]

"**FR7 View Map**: The user can view a map that displays their current location and any historic landmarks around them, within a 5 mile radius."

© 2025 Nadine von Frankenberg

Functional Requirements

- Describe the essential functionality of the system
- Define one requirement per function (feature)
- They should be phrased in a general manner



[Example]

"**FR1 Select location**: The user can select a button to zoom in Google Maps and look at a red and green-colored heat map of popular areas."

What is functionality?

- Describes specific tasks the system performs to achieve an intended goal
  e.g., processing data, user interactions, or system operations
- Focuses on how the system responds to inputs and deliver outputs
- Defines **features and services the user can invoke / system provides**
  e.g., login, view a report, create a report, …

Functionality does not describe qualities, e.g., response time, platform type, etc!

[Example]

- Functionality: "The user can upload files."
- Quality attributes
  - "The system can upload files with minimal delay of max. 15 seconds, even with the maximum supported file size."
  - "The system can store uploaded files with a size of up to 2GB."
  - "The system supports files of the types .pdf, .jpg, and .txt"

What is functionality? (cont.)

- **Single purpose:** Each functional requirement should represent one logical unit of work (one operation or one step from the actor's/user's perspective)
- **Low complexity:** Complex requirements should be split into separate requirements (or grouped as sub-requirements)
- **Unified testing & maintenance:** Each functional requirement should be treated as one unit during testing and maintenance
- An **atomic requirement** is a requirement that cannot be further broken down into individual tasks

Splitting complex requirements into smaller, manageable requirements helps in system design to better identify reusable parts

[**Example**] Defining requirements

FR3 Login: Students can login to the system using their email address and password.

User's perspective: Single operation/single purpose
Sequential dependency & low complexity
Testing & maintenance will most likely be treated as a single unit

FR4 Control room temperature automatically: The system automatically controls the temperature for each room.

[**Example**] Defining requirements

User's perspective: viewing and downloading an account statement are two distinct actions
Each has its own functionality and does not depend on each other
Testing & maintenance will most likely be treated separately

FR4 Access account statements: The customer can access account statements.

- **FR4.1 View Monthly Statements:** The customer can view their past transactions in their account statement, updated monthly.
- **FR4.2 Download Statements:** The customer can download their statements as a PDF file.
- **FR4.3 Receive statements via email:** The customer can choose to receive their account statement automatically via email automatically each month.

Sub-steps indicate a logical categorization

© 2025 Nadine von Frankenberg

Deriving requirements: FURPS+ Model

- FURPS+ is a model used to categorize features and attributes
- **F**unctional Requirements

What functionalities/features should the system provide?

- Non-functional Requirements
  - **U**sability
  - **R**eliability
  - **P**erformance
  - **S**upportability
  - …

E.g., robustness, maintainability, …

- Constraints (pseudo-requirements)

E.g., standards, physical limitations, software limitations, dependencies, …

Abbott's textual analysis (technique)

- Methodology used to identify and specify objects in a software system based on natural language
- Review the natural language requirements / problem statement:
  1. **Identify nouns** → actors / objects (or entities)
  2. **Group and categorize** the nouns → define the objects' tasks/services
  3. **Determine the relationships** between the identified objects
     1. Adjectives → describe characteristics/attributes of objects
     2. Verbs → correspond to methods or operations that objects perform
     3. Relationships → describes how objects interact

Introduced by Russell J. Abbott (1983) as a way to systematically extract objects, attributes, and behavior from an (informal) textual description of a system

© 2025 Nadine von Frankenberg

Actor vs entity

- **Actor** is anyone who interacts with the system from outside its boundaries
  - **[Example]**
    Customer who browses and purchases items
    Payment Processing System (external system)
- **Entity** is a core object or concept within the boundaries of the system
  - **[Example]**
    Order, Product, Customer within an e-commerce system
    e.g., system tracks Products (e.g., name, price, inventory count)

[Example] Abbott’s technique

Homeowners want a system to control the temperature in different rooms of a house. The system should:

- Keep track of the temperature in each room
- Adjust heating or cooling to keep the rooms comfortable
- Set different temperatures for each room
- Be controlled through a mobile app
- Send an alert if a room's temperature goes too far from the set point

*simplified

[Example] Abbott's technique — Identifying Actors

Homeowners want a system to control the temperature in different rooms of a house. The **system** should:

- Keep track of the temperature in each room
- Adjust heating or cooling to keep the rooms comfortable
- Set different temperatures for each room
- Be controlled through a mobile app
- Send an alert if a room's temperature goes too far from the set point

Relevant **Actors**:

- (End)*user* = *Homeowner* or *Occupant*
- *Home* (or *Heating and cooling system*)
- *System* (to be built)

An actor is an entity that interacts with a system

At this stage, the system to be built can be seen as an actor, as the system boundaries still need to be defined.

*simplified

[Example] Abbott’s technique — Identifying Actors

University of Pittsburgh

Homeowners want a system to control the temperature in different rooms of a house. The system should:

- **Keep track of the temperature** in each room
- **Adjust heating or cooling** to keep the rooms comfortable
- **Set different temperatures** for each room
- Be controlled through a mobile app
- **Send an alert** if a room's temperature goes too far from the set point

*simplified

[Example] Abbott's technique — Identifying Actors

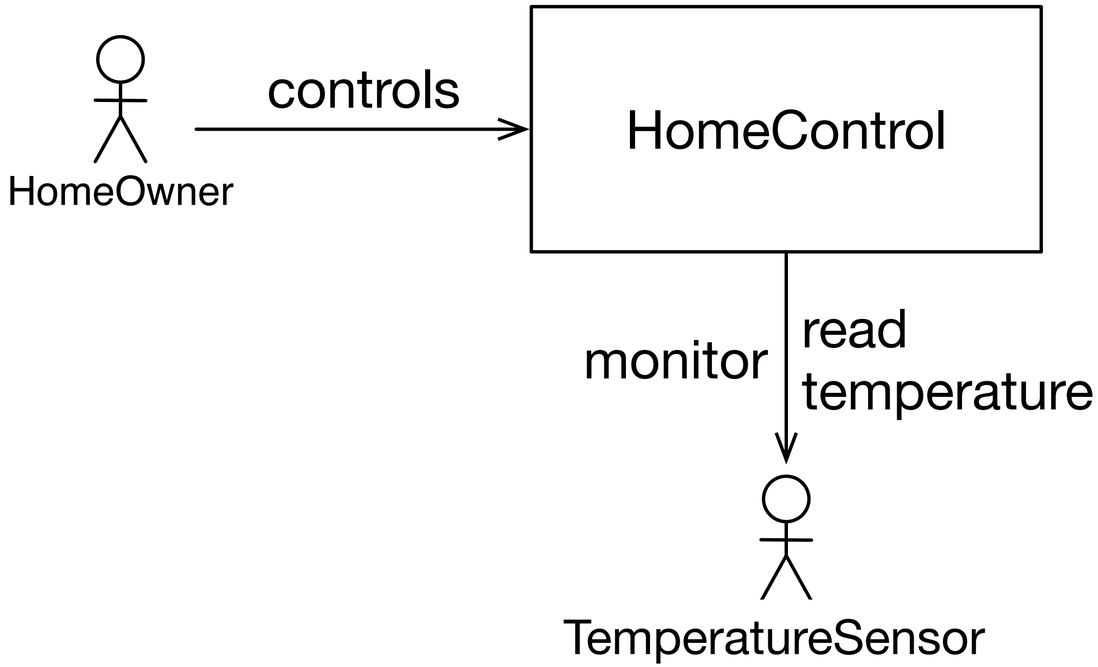
Homeowners want a system to control the temperature in different rooms of a house. The system should:

- **Keep track of the temperature** in each room
- **Adjust heating or cooling** to keep the rooms comfortable
- **Set different temperatures** for each room
- Be controlled through a mobile app
- **Send an alert** if a room's temperature goes too far from the set point

Actors/*Entities*:

- **HomeOwner**: View and set temperatures, receive alerts
- **HeatingCoolingSystem**: Change the temperature in the rooms
- TemperatureSensor: Detect the temperature in each room

[Example] Abbott's technique — Identifying Actors



© 2025 Nadine von Frankenberg

[**Example**] Abbott's technique

*simplified

University of Pittsburgh

The enduser or user of the system is defined as the home owner or occupant.

FR1 View room temperature: The user can view the current temperature of each room.

FR2 Monitor room temperature: The system monitors the temperature of each room.

FR2.1 Sense temperature: The temperature sensor senses the room's current temperature.

FR2.2 Read temperature: The system reads the temperature sensor's temperature readings once a minute.

FR3 Control room temperature manually: The user can control the temperature for each room.

FR4 Control room temperature automatically: The system automatically controls the temperature for each room.

FR5 Send temperature alerts: The system sends alerts to the user if the temperature in a room deviates from the set preference by more than 4°F.

FR6 Set room temperature preferences: The user can set a desired temperature for each room.

© 2025 Nadine von Frankenberg

[**Example**] Abbott's technique - alternative

*simplified

University of Pittsburgh

The enduser or user of the system is defined as the home owner or occupant.
The software system to be developed is hereafter referred to as HomeControl.

FR1 View room temperature: The user can view the current temperature of each room.

FR2 Monitor room temperature: HomeControl monitors the temperature in each room.

FR2.1 Sense temperature: The temperature sensor senses the room's current temperature.

FR2.2 Read temperature: HomeControl reads the temperature sensor's temperature readings once a minute.

FR3 Control room temperature manually: The user can control the temperature for each room and set it to a specified temperature value.

FR4 Control room temperature automatically: HomeControl automatically controls the temperature for each room and set it to a specified temperature value using a learning-based schedule (→ FR8).

FR5 Send temperature alerts: HomeControl sends alerts to the user if the temperature in a room deviates from the set preference by more than 4°F.

FR6 Set room temperature preferences: The user can set a desired temperature for each room.

© 2025 Nadine von Frankenberg

System vs external system

- **Software system** is a system/component that is being developed
  - Has specified boundaries
- **External system** is a system/component outside of the boundary of the software system being developed that interacts with it
  - E.g., API, database, sensor, hardware, third-party system

Boundaries define what is inside the system (internal components) and what interacts with it from the outside (external actors, systems/components)

© 2025 Nadine von Frankenberg

Problem Statement

Problem Statement

Background

Pittsburgh is facing an increasing number of potholes and sinkholes that pose a threat to public safety and infrastructure. These sudden ground collapses can cause considerable damage to roads, disrupt supply lines, and endanger human lives. Current methods of monitoring and responding to sinkholes are reactive and inefficient. There is an urgent need for a proactive, technology-based solution to detect, track, and predict sinkholes. This solution would enable municipal authorities to respond quickly to potential hazards, deploy resources more efficiently, and take preventative measures to mitigate the impact of sinkholes on the community and infrastructure.

© 2025 Nadine von Frankenberg

Problem Statement

The City of Pittsburgh requires the development of a Sinkhole Monitoring System (SiMCity) to address the issue of sinkholes within the city. The first version of SiMCity should enable residents and city officials to report sinkhole occurrences and track their locations.

The system should be easy to use, accessible through web and mobile platforms, and track sinkholes in real-time.

Residents should be able to view sinkholes on a map, helping them to avoid such areas. It should also allow residents to report new sinkholes by sending in their location and photographic evidence for verification, or call for help. City officials can remove sinkholes from the map after they have been re-filled and repaired.

SiMCity aims to increase public safety, facilitate rapid response to sinkholes, and improve the city's ability to manage and mitigate the impact of these geohazards.

Let's develop this system!

© 2025 Nadine von Frankenberg

I06 — Define functional requirements

⏳ 15 min

👩🏽‍💻👨🏻‍💻 Pairs

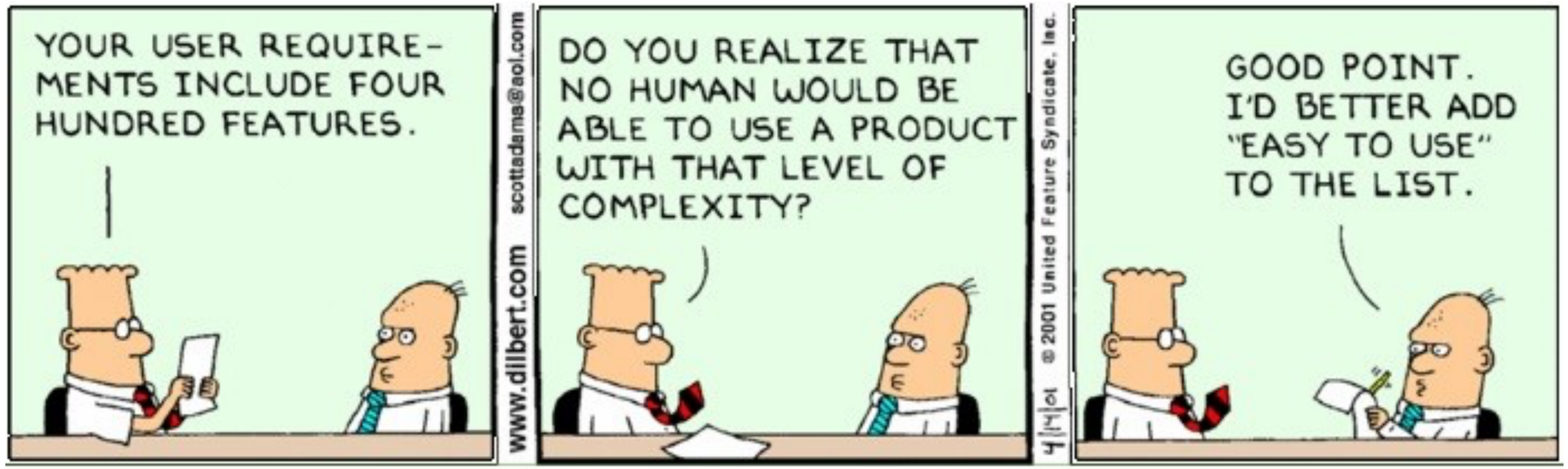
University of Pittsburgh

Based on the problem statement, define **functional requirements** for the SiMCity system

continued in L07!

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

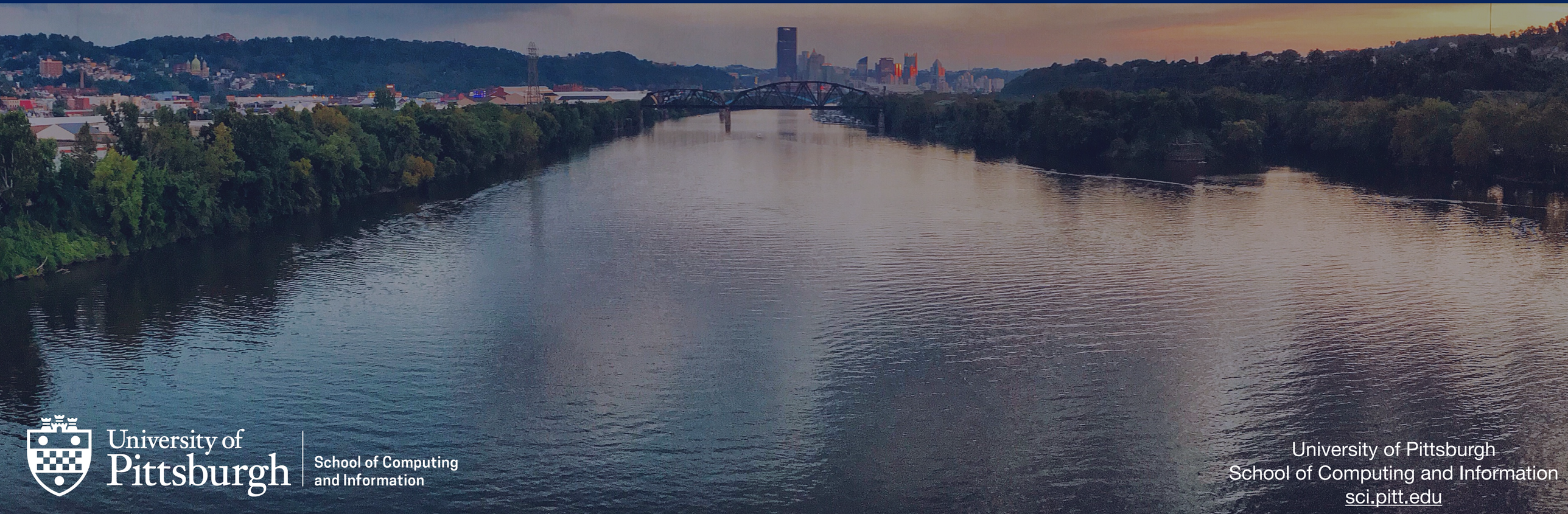


Fall 2025

L06 Requirements Elicitation I

CS 1530 Software Engineering

Nadine von Frankenberg



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