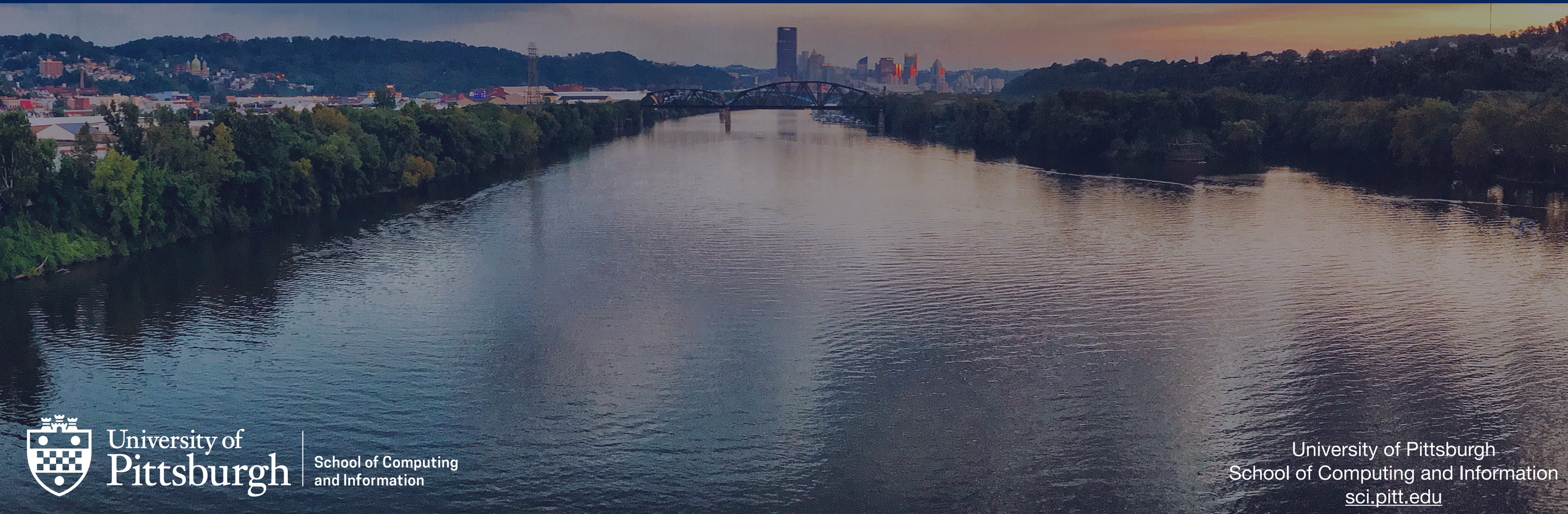
Fall 2025

L07 Requirements Elicitation II

CS 1530 Software Engineering

Nadine von Frankenberg



University of Pittsburgh
School of Computing and Information

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 deeper understanding of functional requirements
- You can define non-functional requirements
- You can define constraints

Today's roadmap

- Deriving requirements
  - [Recap] Deriving functional requirements
  - Non-functional requirements

Goals of Requirements Elicitation

- Define the **problem**: Understanding the stakeholders' real needs (not just what they think they need)
- **Avoid ambiguity**: Requirements should be clear, testable, and unambiguous
- **Build a shared understanding**: Align developers, customers, and other stakeholders
  - Users, endusers, customers, maintainers, support staff, etc.

Proper requirements elicitation avoids rework, saves money, and keeps projects on track.

[Examples] Issues with requirements

- healthcare.gov: Launch in 2013, crashed under high load
- Zoom: Security issues (2020)
- Windows/CrowdStrike (mid 2024): No backward compatibility (BSOD)
- Tesla HW4 Computer Failures (late 2024): Safety requirements inadequate
- …

[Recap] Functional Requirements

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

FR# **Short title** : Description

Should be numbered based on their SOON priority

Phrased in **imperative mood**!

The description should not include implementation details!

[Example]

FR1 Select location: The user can select a location on a map to view sinkholes in the selected area.

FR2 Login: The user can login to the system using their email address and password.

[Recap] What is functionality?

- **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

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.

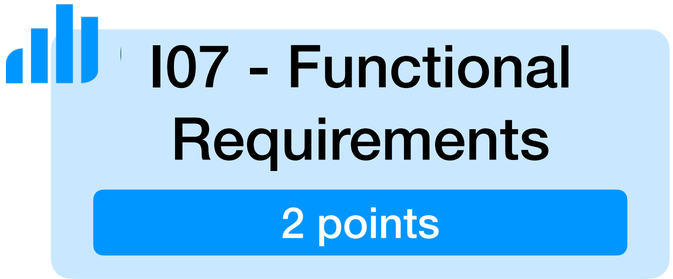
Sub-steps indicate a logical categorization

I06/I07 — Define functional requirements

⏳ 15 min

👩🏽‍💻👨🏻‍💻 Pairs

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



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!

Today's roadmap

- Deriving requirements
  - [Recap] Deriving functional requirements
  - Non-functional requirements

[Recap] Deriving requirements: FURPS+ Model

University of Pittsburgh

- 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)

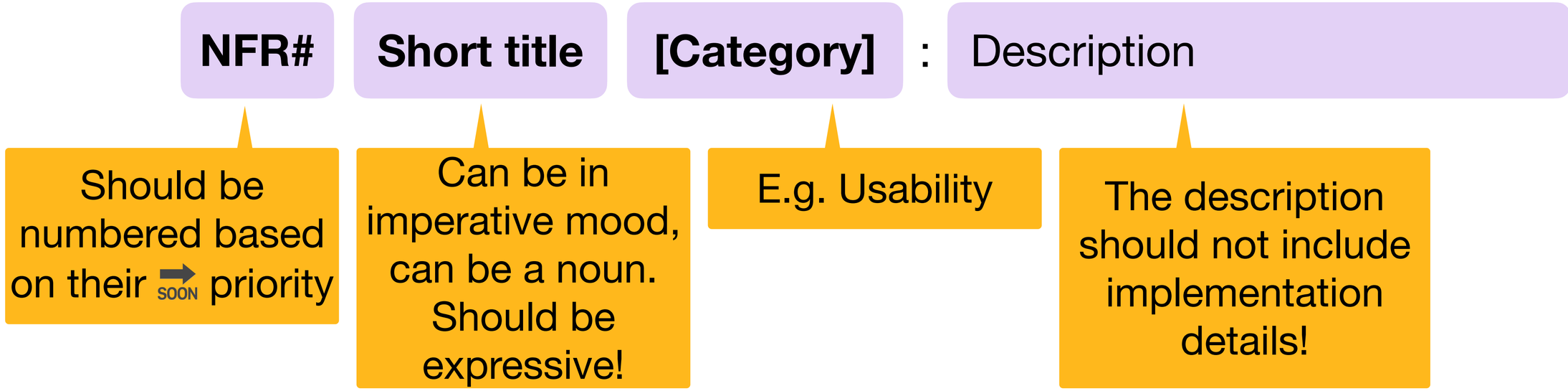
E.g., standards, physical limitations, software limitations, dependencies, …

Non-functional requirements & constraints

- **Non-functional requirements** (NFR) describe quality attributes that are important for the system (but do not describe functionality)
- **Constraints** describe limitations or conditions that restrict the solution space
  - E.g., budgetary constraints, time constraints, technical constraints (e.g., the use of specific technologies), …

Defining non-functional requirements

- **Usability**: How easy to use is the system?
- **Robustness**: How stable and reliable is the system?
- **Performance**: How fast and efficient is the system?
- **Security**: How secure is the system?
- …



Defining non-functional requirements

University of Pittsburgh

- NFRs are observed during the testing, validation & verification phase
- Focus on **measurable**, **testable**, and **specific** (enough) criteria!
  - It is (often) up to the requirements engineer to define **measurable** NFRs
- Problem statements often hint at NFRs

[Example]

"Wording and color scheme should be consistent"

"The user should always be able to get back to the dashboard within at most 2 clicks."

"The system must be user-friendly", "The system should be intuitive", "The system should be easy to use"

University of Pittsburgh

"The system must be

easy to use

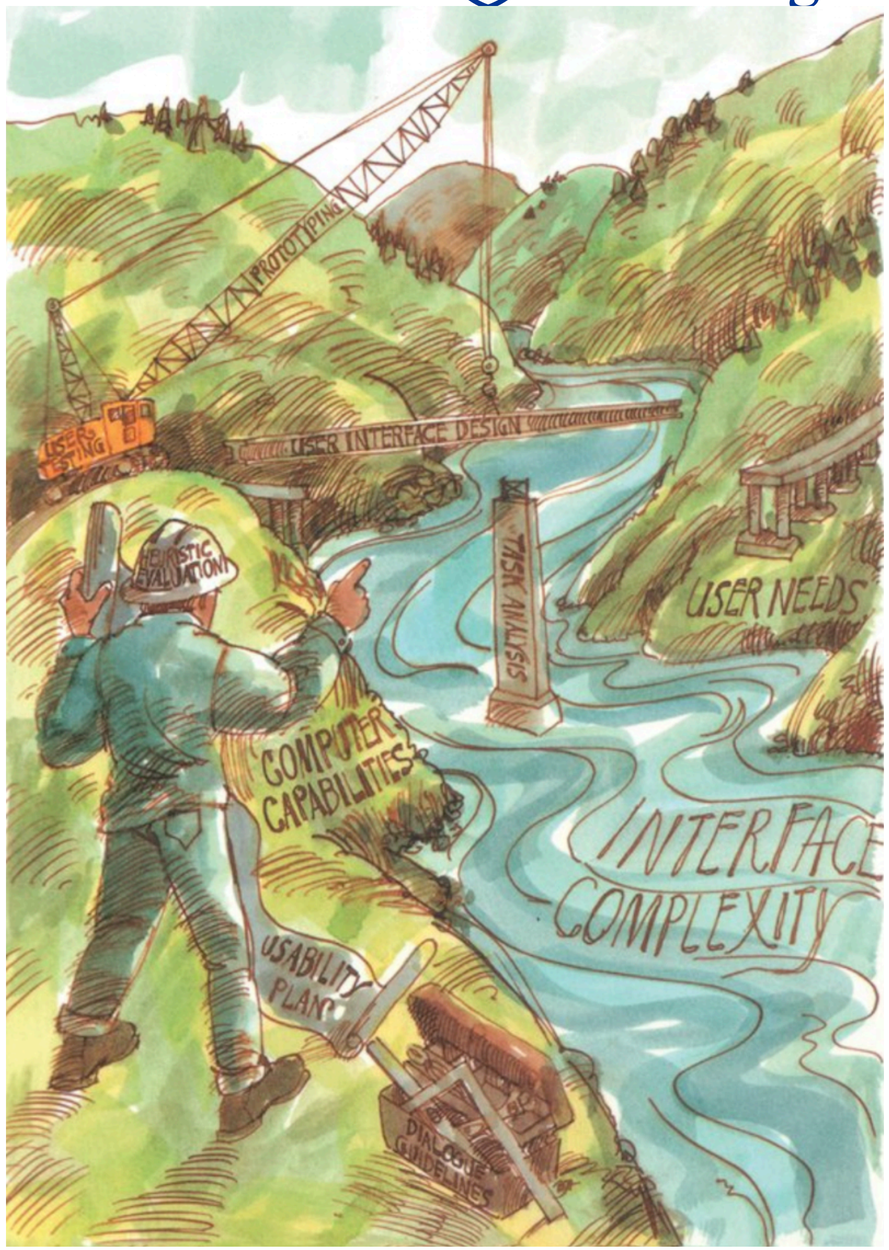
user-friendly

intuitive

...

Usability — Overview

- "*How effectively, efficiently and satisfactorily a user can interact with a user interface*" (usability.gov)
- "*Quality attribute that assesses how easy user interfaces are*" (Nielsen, 2003)



Usability Engineering

- Process to "build" usability into products
- Helps to ensure "high quality" of products

Why is Usability Engineering important?

How can we describe usability?

Usability

- **Learnability** — How quickly can users grasp the user interface?
- **Efficiency** — Once mastered, does the interface enable swift and efficient usage?
- **Memorability** — Can users easily recall how to navigate and use the interface? Is the design memorable and user-friendly over time?
- **Error Handling & Robustness** — How well does the system handle errors or unexpected user actions?
- (subjective) **Satisfaction & User Experience** — Does the user interface offer an enjoyable and satisfying experience?

Usability (cont.)

- **Consistency**: Is the interface design uniform across different sections, reducing cognitive load for users? (7±2) !
- **Flexibility**: Can users customize the interface to better suit their individual preferences and needs? E.g., IDE themes
- **Task Completion**: How effectively does the interface support users in accomplishing their intended tasks without unnecessary complications?
- **Feedback**: Does the interface provide users with helpful feedback after performing an action or guidance when they encounter difficulties or make errors?
- …

[Example] Usability requirements

NFR1 Student Learnability [Usability]: Students must be able to use the system without needing a manual.

NFR2 Instructor Learnability [Usability]: Instructors must be able to learn and recall the main functionality of the system after following a user tutorial.

[Example] Usability requirements

NFR1 Student Learnability [Usability]: Students must be able to use the system to complete basic tasks within 10 minutes without requiring a manual.

NFR2 Instructor Learnability [Usability]: Instructors must be able to learn and recall 90% of the core system functionality after completing a one-hour tutorial and must be able to perform key tasks within 5 minutes after a one-week break.

[Example] Usability requirements

Initial version. Measurability is vague, should be revised and refined when more is known

NFR1 Student Learnability [Usability]: Students must be able to use the system without needing a manual.

NFR2 Instructor Learnability [Usability]: Instructors must be able to learn and recall the main functionality of the system after following a user tutorial.

Revised / more detailed version

NFR1 Student Learnability [Usability]: Students must be able to use the system to complete basic tasks within 10 minutes without requiring a manual.

NFR2 Instructor Learnability [Usability]: Instructors must be able to learn and recall 90% of the core system functionality after completing a one-hour tutorial and must be able to perform key tasks within 5 minutes after a one-week break.

Today's roadmap

- Deriving requirements
  - [Recap] Deriving functional requirements
  - Non-functional requirements
- Organizational: Team Project

[Team Project]

- **Team**: 4 - 6 students
- **Goal**: Develop a software system throughout this term
  - Learn to apply SE best practices
  - Become more confident in using git and build up your GitHub profile
  - Setup a development pipeline and follow a workflow
  - Create test cases
  - Learn to use new tools
  - …

[Team Project] Timeline & First Tasks

- Team forming: Due **September 18, 2025, 11:59pm**
- Problem statement: Due **September 25, 2025, 11:59pm**
- Find someone (your customer) who has a **problem** that **can be solved** with a (somewhat) complex **software system**
  - Customers can be:
    - Family members, friends
    - Pitt community: Professor (me included), staff
    - Local shop owner
    - …

!! If you already have an idea, please make use of Discord to find a team by 09/18.
Otherwise I will allocate the teams.

- System examples:
  - Campus navigation app
  - Fitness challenge platform
  - Personal finance tracker
  - Smart grocery list
  - Local events website
  - Sustainability tipps app
  - Coding challenge website
  - …

[Team Project] Timeline & First Tasks

University of Pittsburgh

- Team forming: Due **September 18, 2025, 11:59pm**
- If you **already have a team**, you must register it **-> Q2**
  - Name + email + GitHub handle
  - Roles: PO & Scrum Master (can rotate later)
- **If you do not have a team -> Q1**
  - Name + Email + GitHub handle
  - Preferred Role: (PO, SM, Developer — or "no preference", can be changed later)
  - Project interests (short text or category you find interersting), no guarantee though 🙃

!! If you already have a project idea but no team / not enough members, please make use of Discord to find a team by 09/17,
otherwise I will allocate you into teams.

[Team Project] Basic requirements

- Project must be hosted open-source (GitHub or GitLab) (more on that soon!)
- Project must contain:
  - User interface
  - Middleware (logical components)
  - Data storage
  - *Optional*: connect to an external API

- Meet at least on a (bi-) weekly basis
- Work as a TEAM (you are all responsible for the outcome!)
  - Each major milestone will include a peer review of the other team members.

Team project schedule *[tentative]*

University of Pittsburgh

#	Milestone	Pt	Start	Due	Format
-	Team formation	1	09/16	09/18	Team / preferences
0	Project Kickoff (Problem Statement)	5	09/19	09/25	Problem Statement
1	Sprint 1: Requirements & Use Cases	15	09/25	10/07	Requirements, user stories, use case diagram, acceptance criteria
2	Sprint 2: Analysis & Modeling	10	10/07	10/21	Analysis object model, class diagram, 2 dynamic models
3	Sprint 3: Architecture & Tech Stack	10	10/21	11/04	Architecture model, choice of stack
4	Sprint 4: Implementation	0	11/04	11/18	Basic skeleton running
5	Sprint 5: Presentation & Delivery	50	11/18	12/05	Final code, documentation, live demo

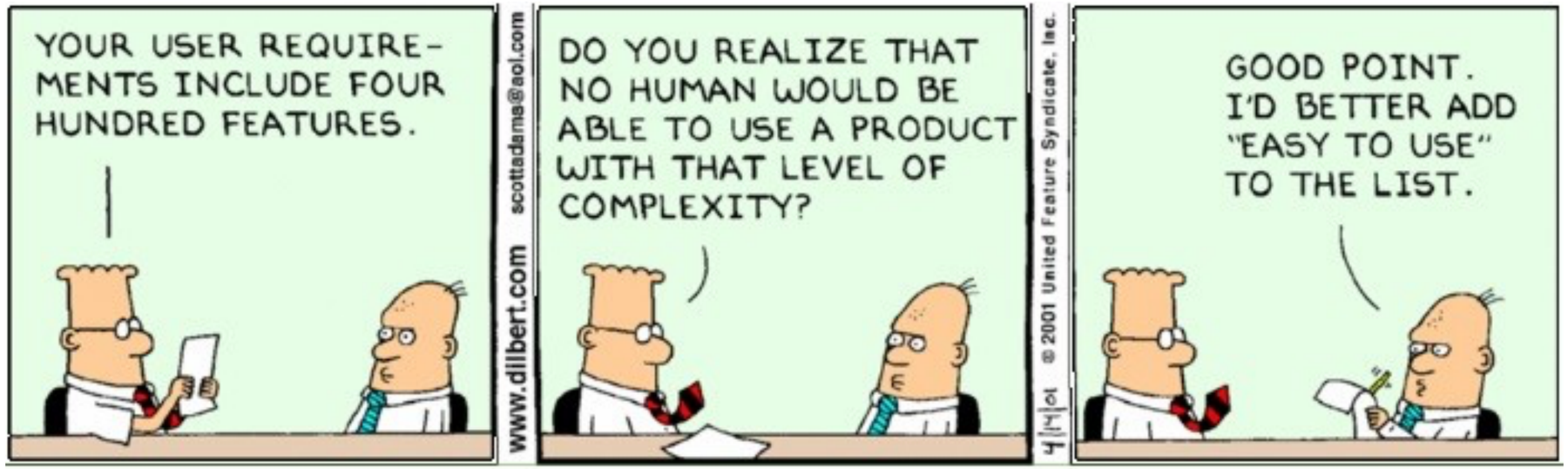
Homework assignment schedule *[tentative]*

University of Pittsburgh

Assignment	Pt	Start	Due	Topic
A1	25	09/18	09/27	Requirements
A2	25	10/07	10/16	Modeling
A3	25	10/30	11/11	Patterns
A4	25	11/18	12/05	Testing

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

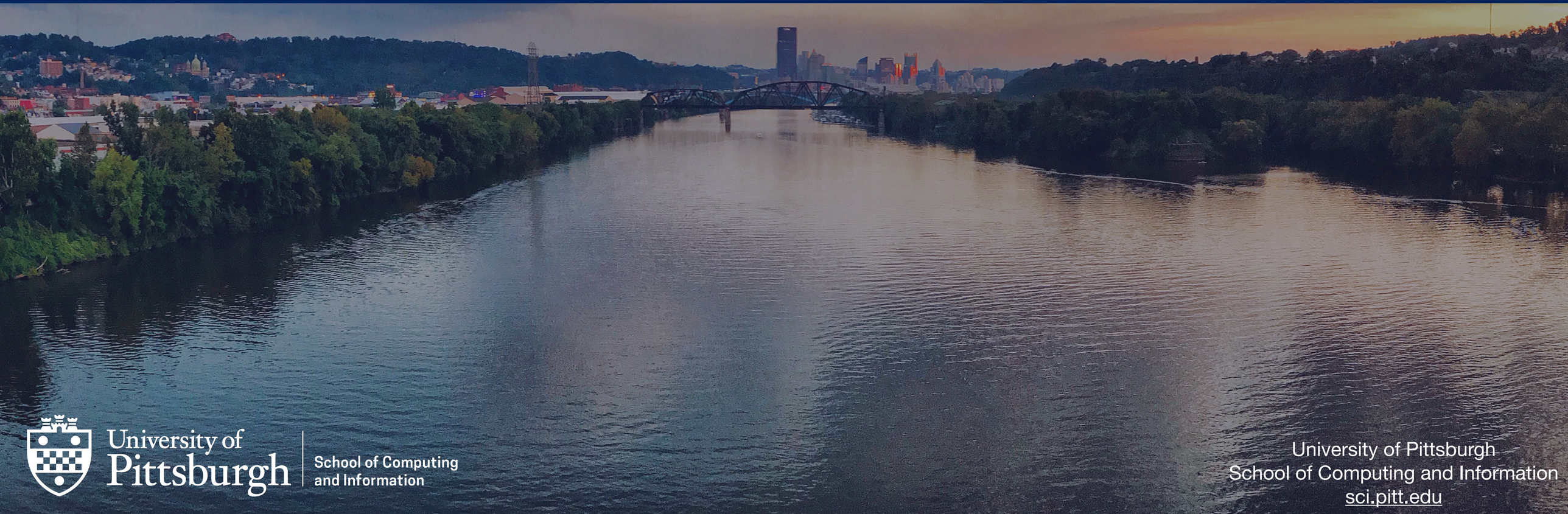


Fall 2025

L07 Requirements Elicitation II

CS 1530 Software Engineering

Nadine von Frankenberg



University of Pittsburgh
School of Computing and Information

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