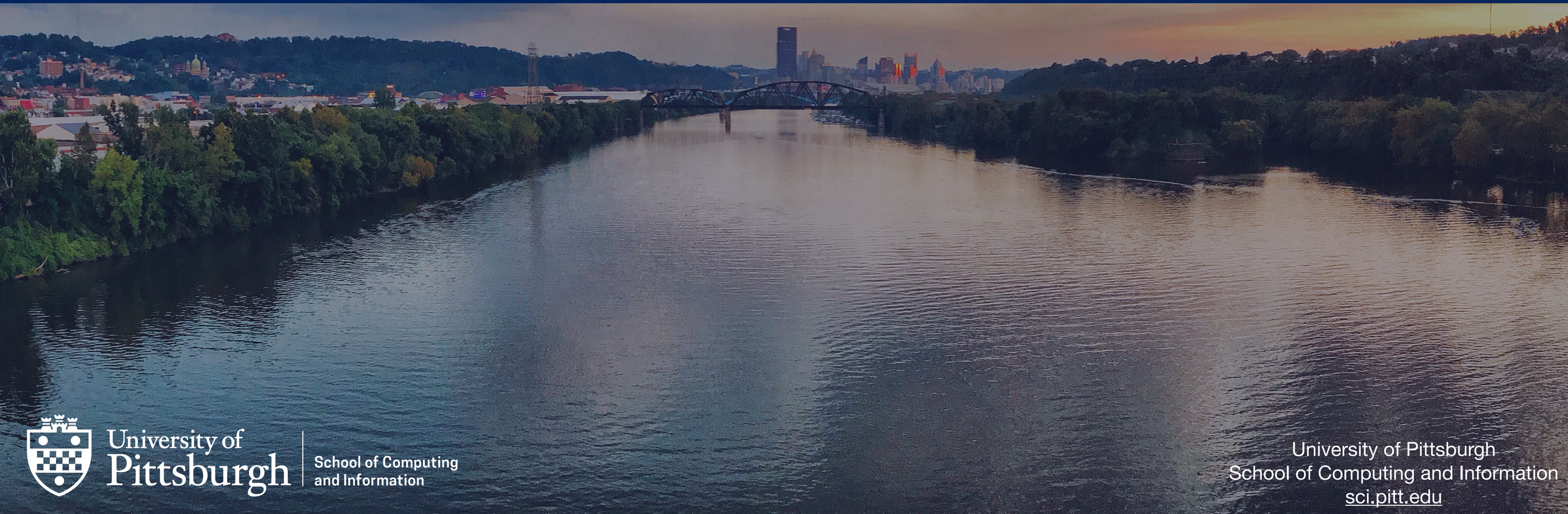
Fall 2025

L02 Introduction to SE

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 an overview of software engineering practices
- You understand the need for process control when building software
- You understand the differences between incremental, iterative, and adaptive development approaches

Today's roadmap

- A short history of SE
- Intro to software development lifecycle models
  - AdHoc & Waterfall Model
  - Development Approaches
  - Agile Manifesto

Software systems

- Many types of software systems exist:
  - Simple embedded systems
  - Complex search engines
  - Graphics-intensive games
  - …
- There is no common denominator/universal software to build them

All these systems need software engineering!

Software engineering

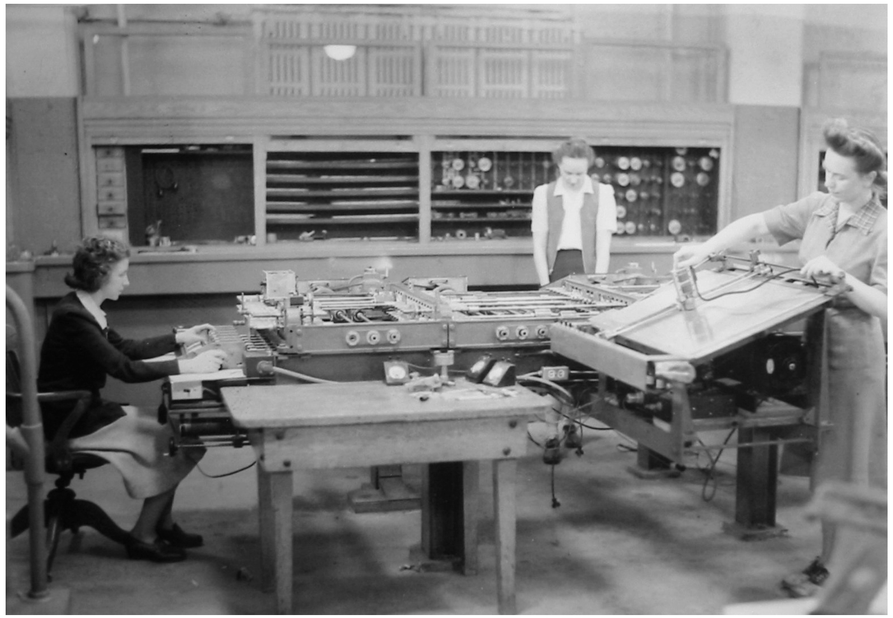
University of Pittsburgh

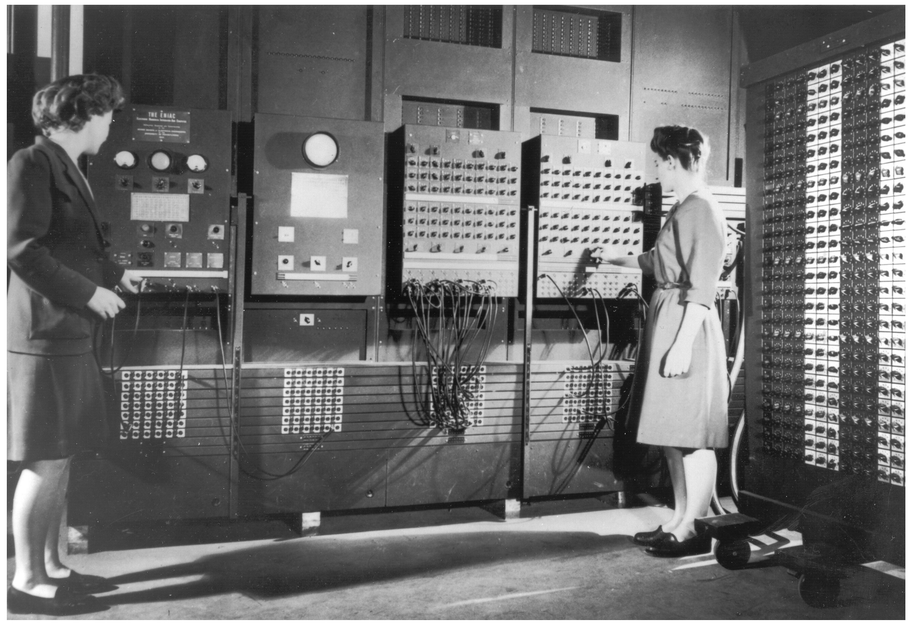
> *Software engineering is an **engineering discipline** that is concerned with **all aspects of software production** from the early stages of system specification through to maintaining the system after it has gone into use.*
>
> - Ian Sommerville

- SE is a **process** concerned with all aspects of producing software to solve real-world problems: specifying, designing, developing, analyzing, testing, deploying, and maintaining software systems
- It needs to adhere to a set of engineering **principles** and **best practices**
- **Involves both technical and non-technical skills**

Software engineering eras

- "Early days" ~1945 — 1950
  - ENIAC
    University of Pennsylvania, 1943 — 1946



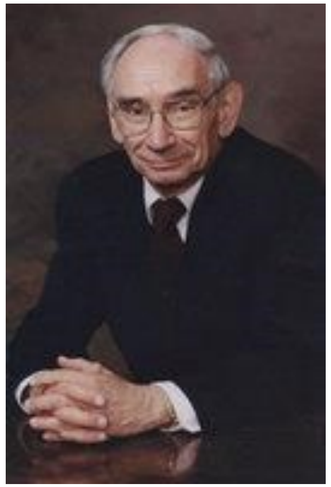


https://penntoday.upenn.edu/news/eniacs-anniversary-nod-its-female-computers

Software engineering eras

- "Early days" ~1945 — 1950
  - ENIAC
    University of Pennsylvania, 1943 — 1946
- **Software crisis** ~mid 1960s — mid 1980s
  - Very large software projects
  - Start of large commercial software applications
  - E.g., IBM System/360, FORTRAN, COBOL, Apollo
  - Margaret E. Hamilton & Anthony Oettinger coined the term "Software Engineering"

Margaret E. Hamilton
Born: 1936
Developed software for NASA's Apollo program
Founded two companies
Presidential Medal of Freedom



Anthony Oettinger
1929 - 2022
Worked on information resources policy (Harvard)
Consulted on NASA's Apollo moon-landing program

The Software Crisis — Causes

- Complexity of software systems
- Increasing demand
- Projects ran over-budget & -time
- Low quality software
- No standards
- Lack of formalization
- Unmanageable, difficult to maintain

1st NATO Software Engineering Conference (1968)
Garmisch-Partenkirchen, Germany

"The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem." — **Edsger Dijkstra** (1972, *The Humble Programmer*)

The Software Crisis — Approaches

- Reduce complexity
- Improve communication
- Introduce standards
- CASE tools (computer-aided software engineering)
- Define processes
- Model-based engineering
- Object-oriented programming
- …

The Software Crisis — Today

- There has been some progress, but some problems persist
  - Schedule / cost estimates are inaccurate
  - Fast pace
  - Productivity of developers does not keep up with demand
  - Customer expectations are not met
  - ….

Still a need for "engineering" the software process!

Today's roadmap

- A short history of SE
- Intro to software development lifecycle models
  - AdHoc & Waterfall Model
  - Development Approaches
  - Agile Manifesto

Challenges in software development

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

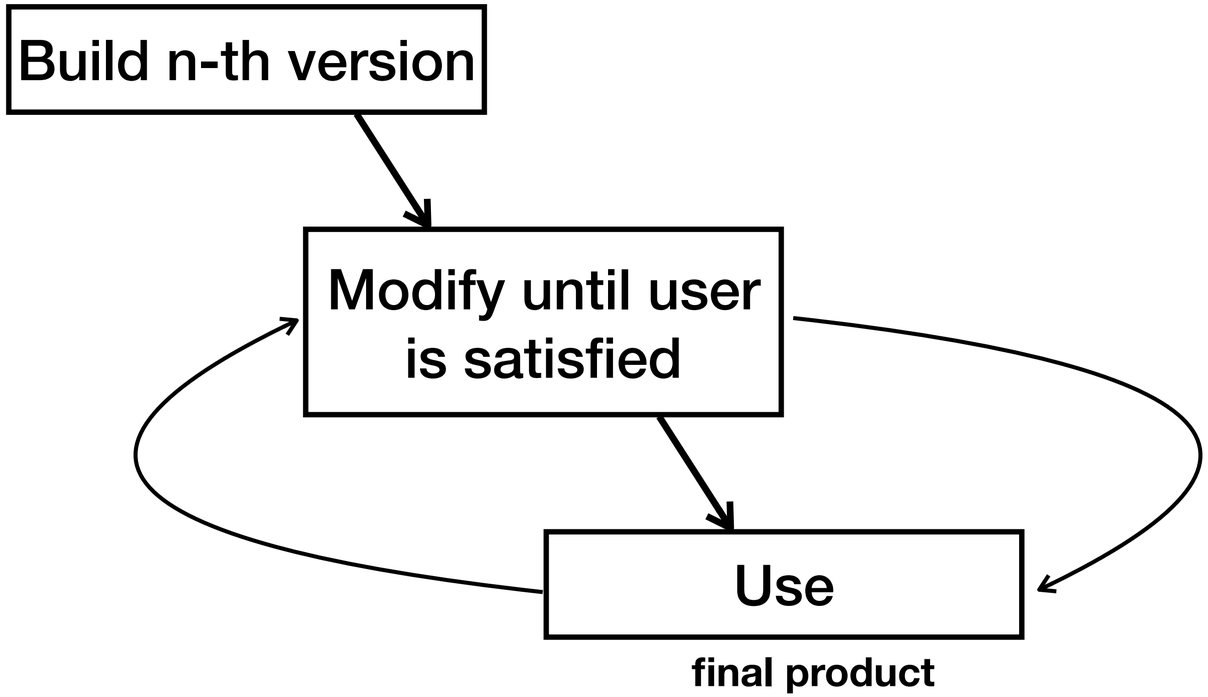
Need for a **process**!

What is a process?

- **Series of steps** involving activities that take some **input**, constraints, and resources to produce an intended **output**
- Involves a **set of techniques and tools**

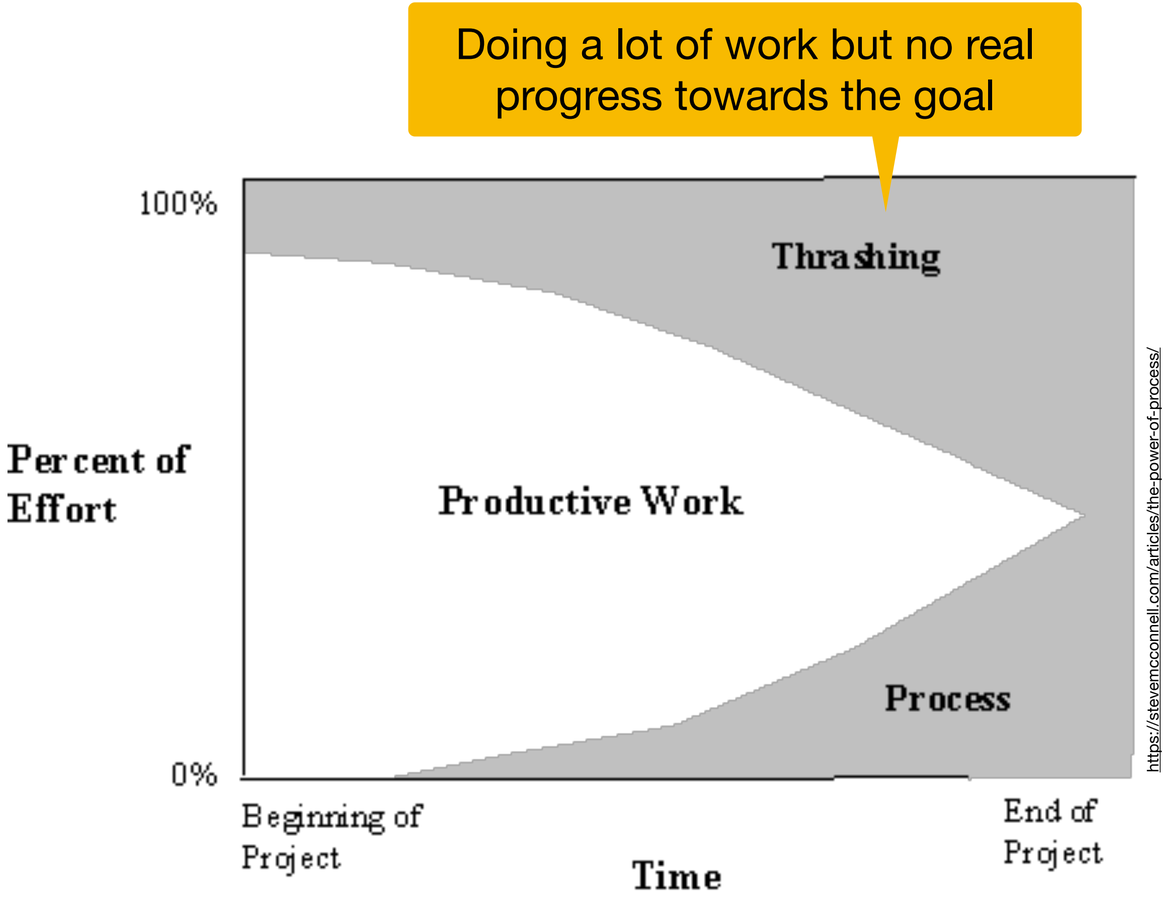
Code & Fix (**Ad hoc Model**)

- Aka. "*Build & fix model*"
- The product is constructed with minimal requirements, minimal or no specification, no attempt at design, and omitting testing
- An initial product is built and then repeatedly modified until the user is satisfied



Ad hoc Model

- Pros
  - Little overhead
  - Quick progress
  - Useful for small, exploratory projects
  - Useful for small teams
- Cons
  - No proper quality assessment
  - Work synchronization is not defined
  - Changes may be hard to incorporate
  - Unclear delivery and timing
  - …



[Excursion] Let's build a house!

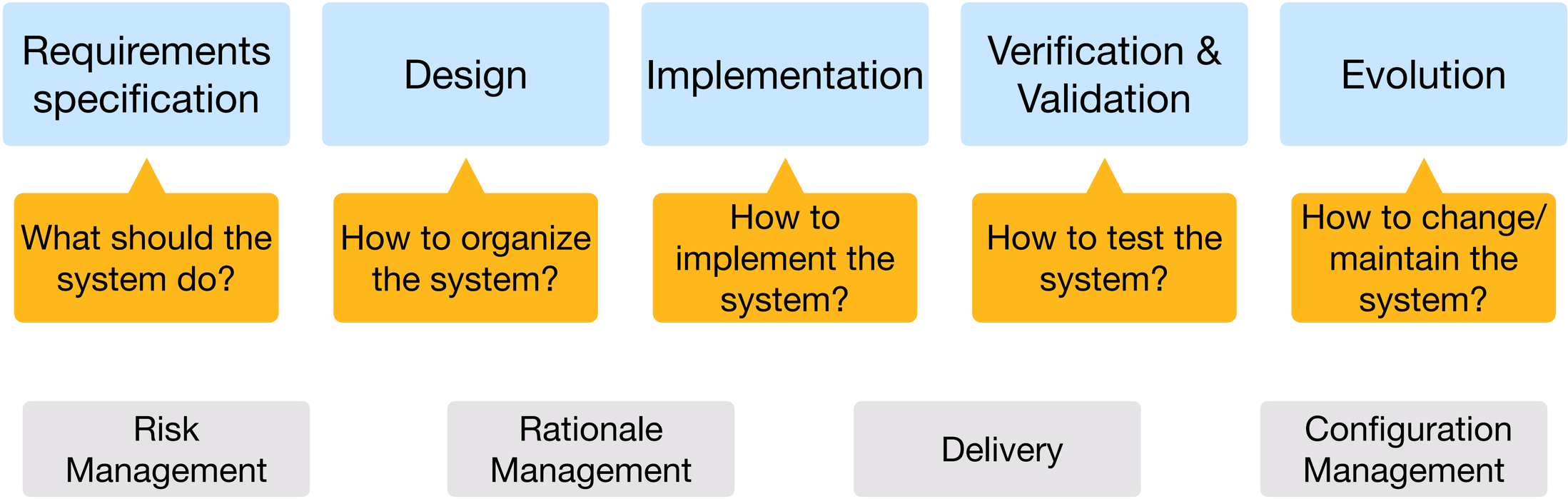
University of Pittsburgh

• Blueprint and requirements	Project Plan & Requirements
• Foundation and architecture	System Design / Software Architecture
• Construction phases	Development (Object Design)
• Materials	Codebase, Tool Suite
• Quality control and inspection	Testing, Deployment/Delivery
• Interior design & occupant experience	User Interaction & Experience
• Maintenance	Updates, Fixes, …
• Collaboration and Teamwork	Developers, Designers, Testers, Stakeholders,…

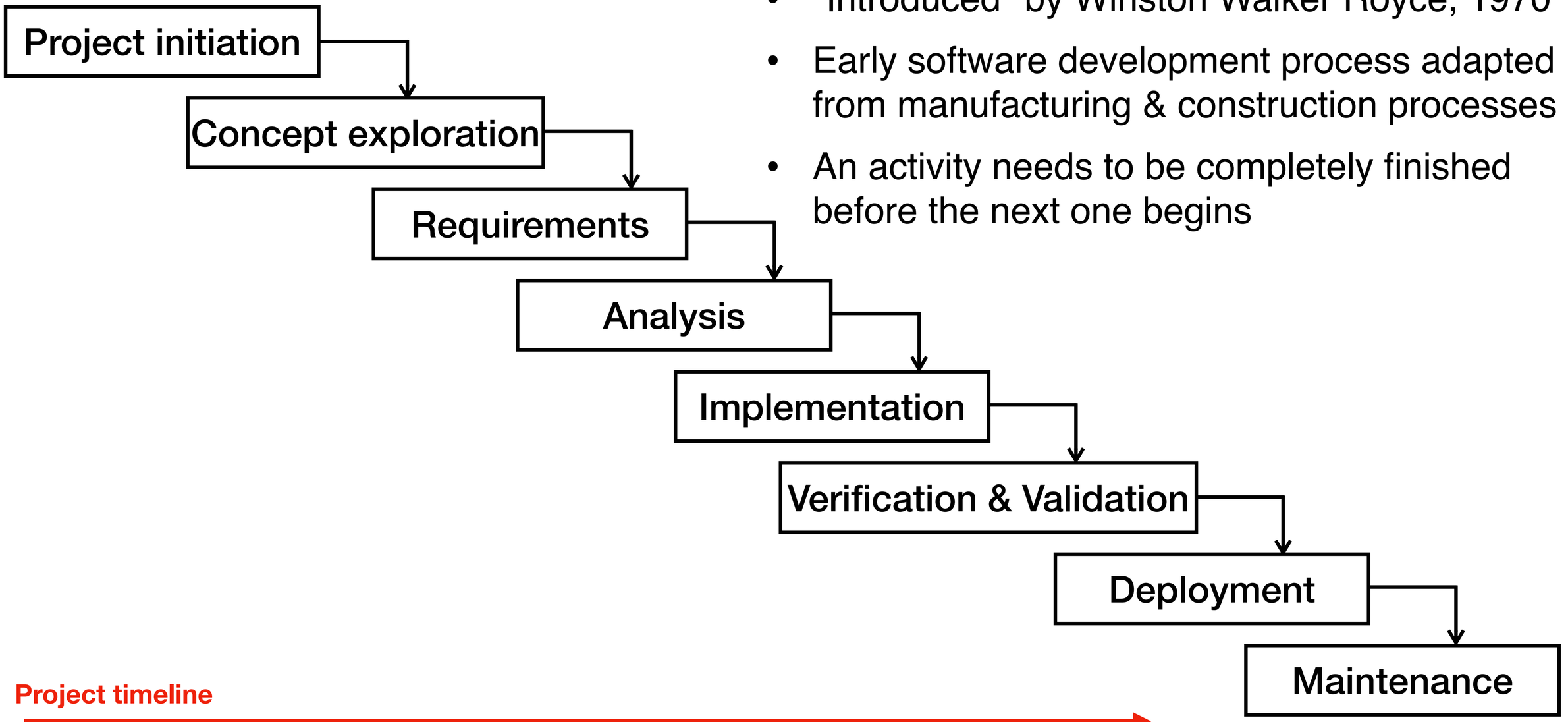
These phases are similar to building a software system!

The software development process

- Structured set of activities required to develop a software system
- There are many different software processes!
- Recurring stages:



The Waterfall Model



- "Introduced" by Winston Walker Royce, 1970
- Early software development process adapted from manufacturing & construction processes
- An activity needs to be completely finished before the next one begins

*exemplary, simplified steps

The Waterfall Model — Considerations

University of Pittsburgh

- Activity-centered lifecycle model
  - Linear and sequential execution of activities (development & management process)
  - All requirements activities are completed before system design activities
  - Goal: Never turn back
- Simplistic view of software development
  - Progress is measured by the number of completed tasks
- No overlapping or iterative activities
- Schedules software development as a step-by-step process
  - Transforms user needs into code
- The verification activity ensures that each activity does not introduce unwanted or delete mandatory requirements
- **Suited for short, simple projects with well-understood requirements**

Today’s roadmap

- A short history of SE
- Intro to software development lifecycle models
  - AdHoc & Waterfall Model
  - Development Approaches
  - Agile Manifesto

What is change in software projects?

- Deviation from initial project plan
  - Evolving requirements throughout the project
- New insights
- Stakeholder feedback
- Challenges
  - Some problems are easier to solve than others
  - Environment
- …

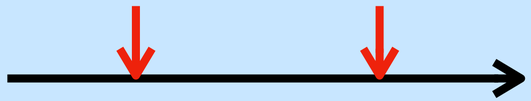
Software lifecycle models & change

- Software lifecycle models vary in their ability to address change
  - **Flexibility**: How can change be addressed?
  - **Responsiveness**: When can change be addressed?
- Lifecycle model choice impacts its ability to handle the frequency of change
- Frequency of change describes how often change occurs, depends on many factors

Rough estimation of change frequency

University of Pittsburgh

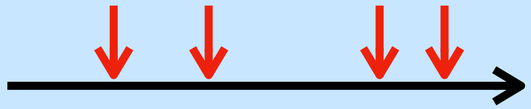
Rarely



→ **Sequential models**

Waterfall, V-Model

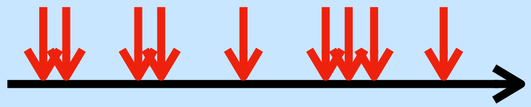
Sometimes



→ **Iterative models**

Spiral Model, Unified Process

Frequently

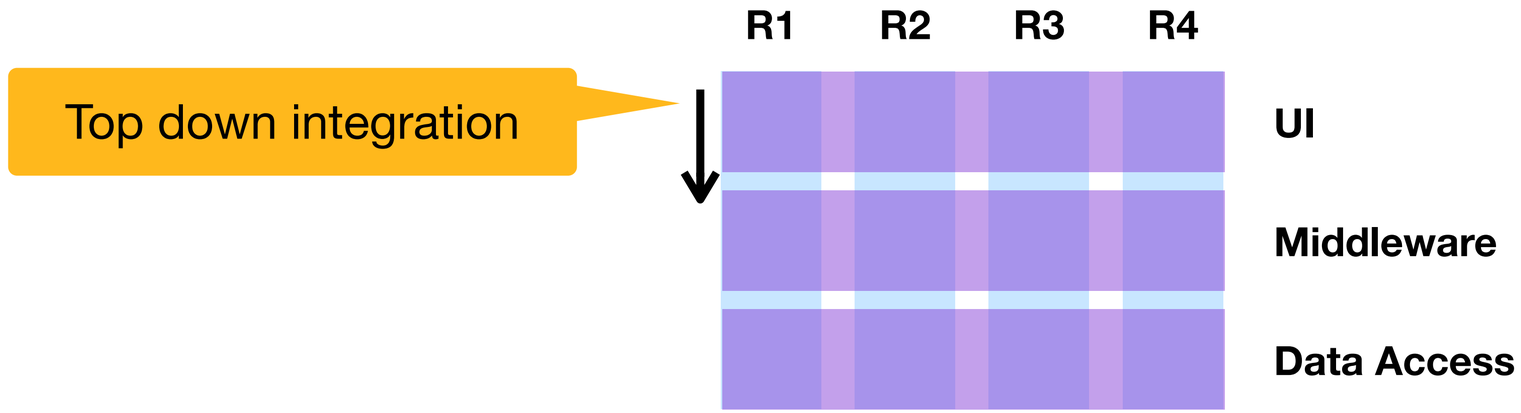


→ **Agile**

Scrum, Kanban, XP

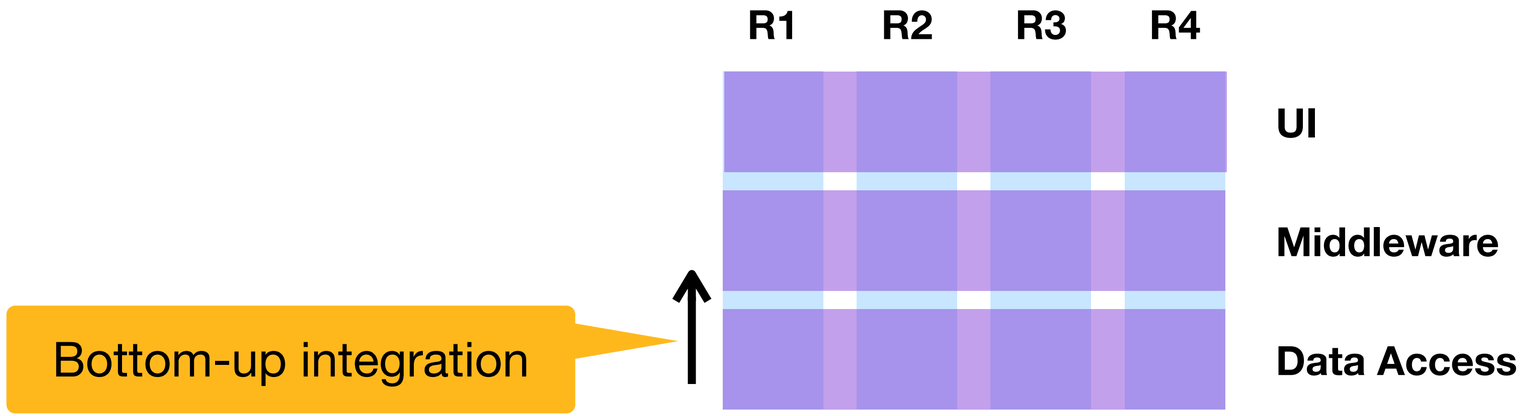
Development approaches

- **Incremental** development (**add on to something)**
  - Build the system in small, usable sections
  - Each increment adds a defined set of functionalities
  - Works well when requirements are well-understood



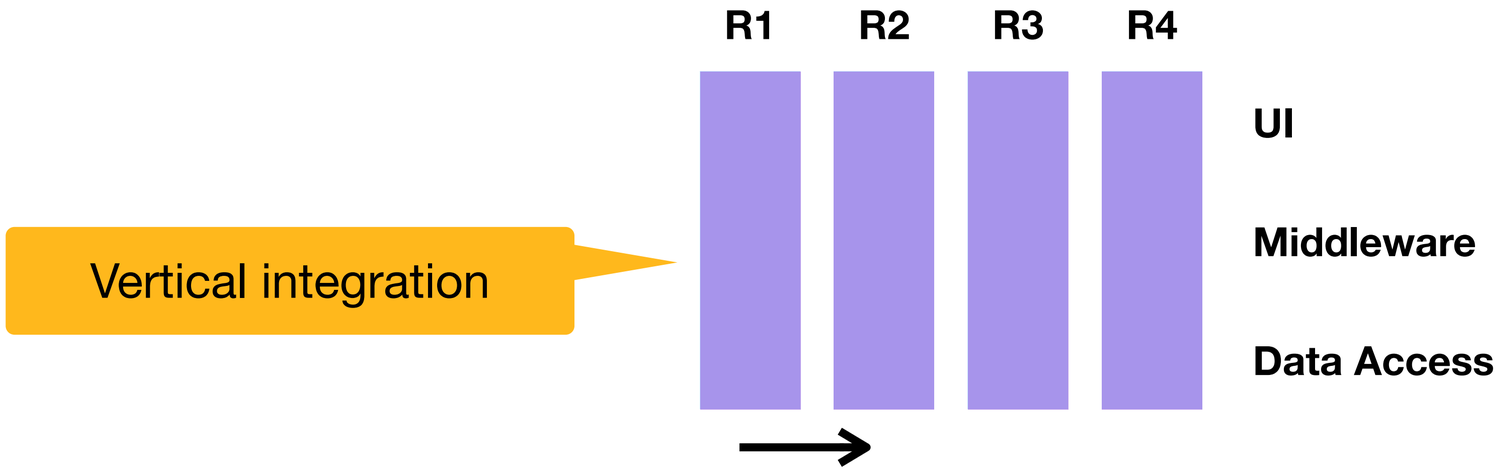
Development approaches

- **Incremental** development (**add on to something)**
  - Build the system in small, usable sections
  - Each increment adds a defined set of functionalities
  - Works well when requirements are well-understood



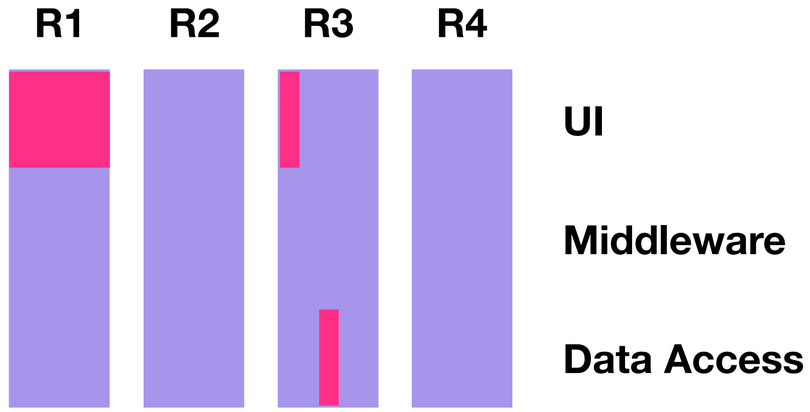
Development approaches

- **Incremental** development (**add on to something)**
  - Build the system in small, usable sections
  - Each increment adds a defined set of functionalities
  - Works well when requirements are well-understood



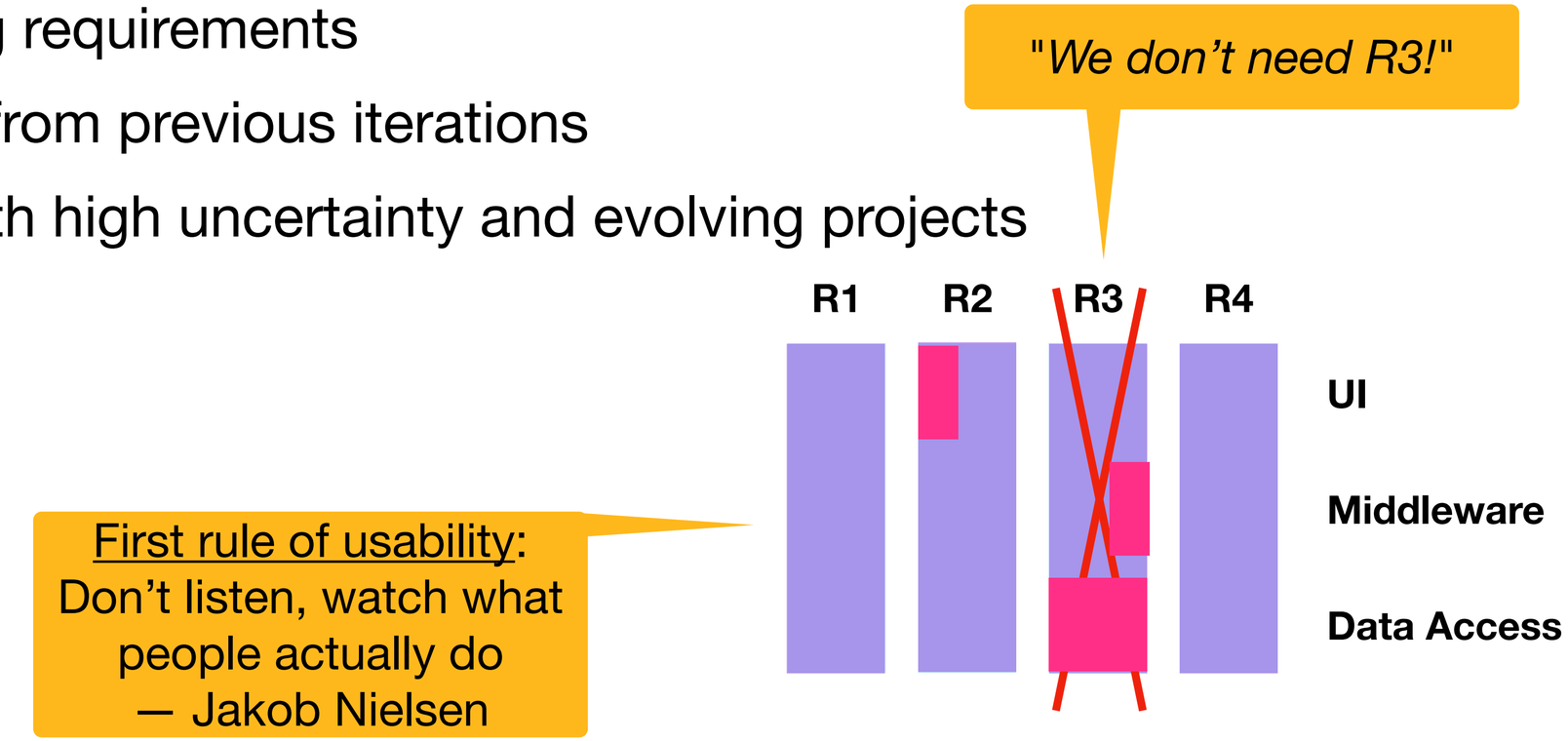
Development approaches

- **Incremental** development (**add on to something)**
- **Iterative** development (**re-do something**)
  - Repeat development cycles on the same functionalities
  - Each cycle improves and refines the product (prototype)
  - Allows to incorporate stakeholder feedback

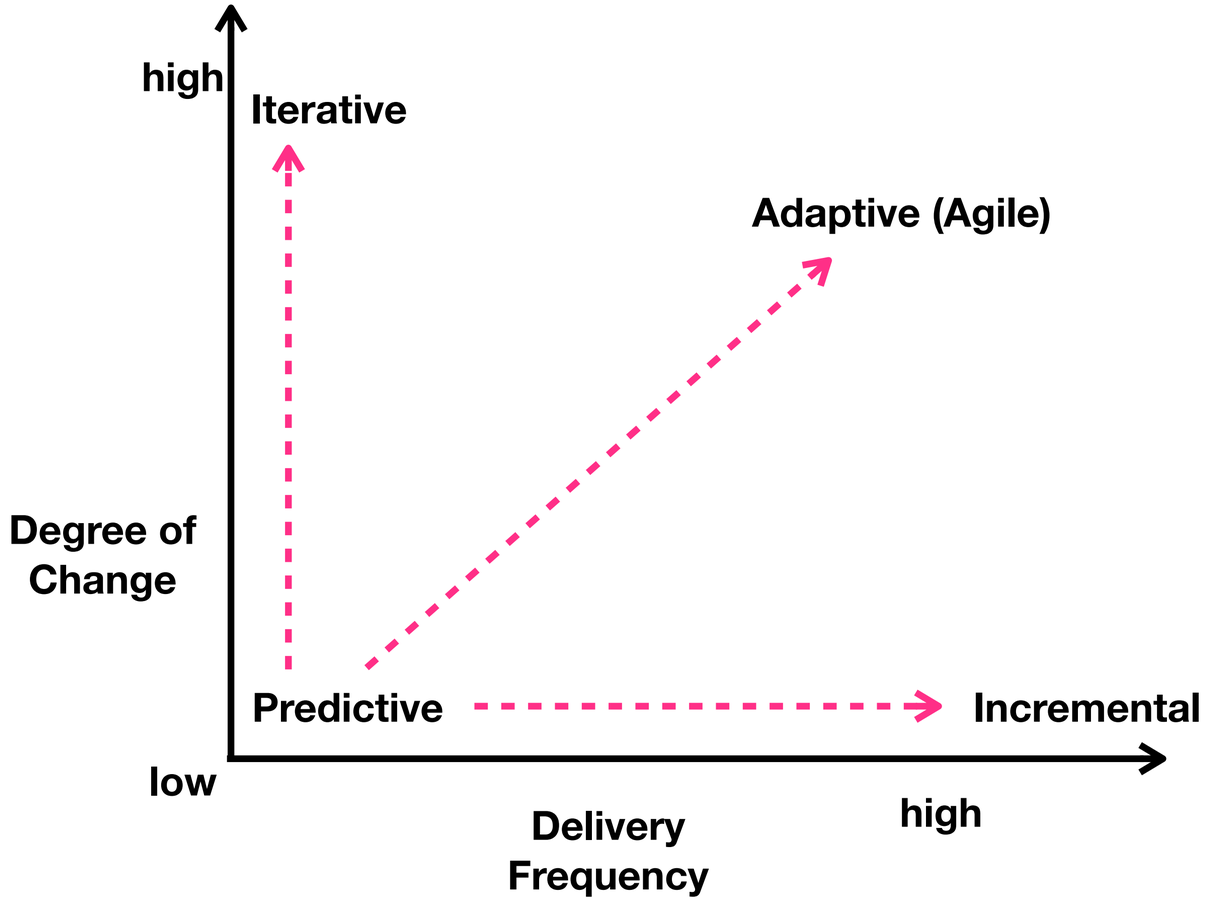


Development approaches

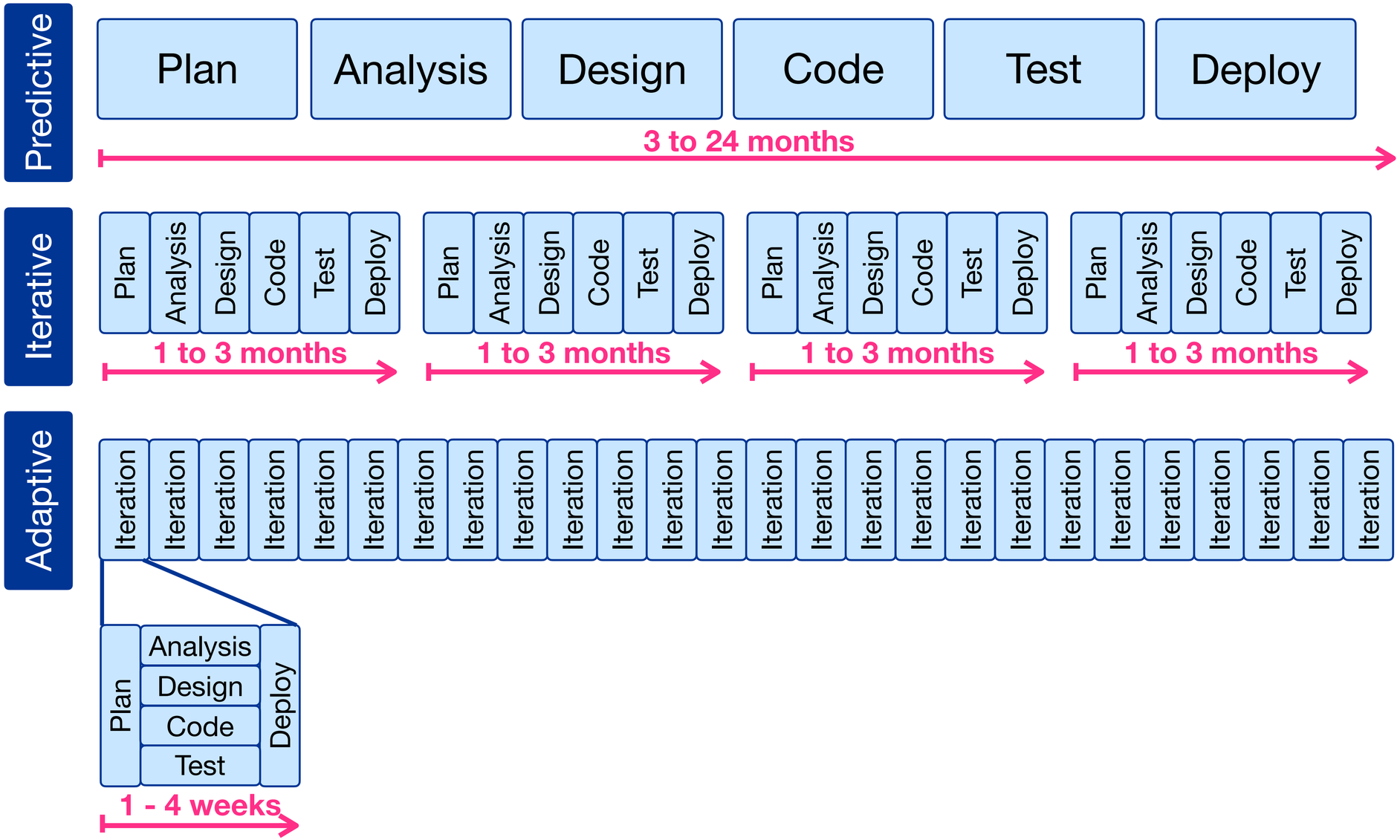
- **Incremental** development (**add on to something)**
- **Iterative** development (**re-do something**)
- **Adaptive** development (**adjust to change**)
  - Highly flexible to changing requirements
  - Adjust based on learning from previous iterations
  - Works well for projects with high uncertainty and evolving projects



Development approaches



Development approaches



Today's roadmap

- A short history of SE
- Intro to software development lifecycle models
  - AdHoc & Waterfall Model
  - Development Approaches
  - Agile Manifesto

Agile manifesto (2001)

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions *over processes and tools*
Working software *over comprehensive documentation*
Customer collaboration *over contract negotiation*
Responding to change *over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

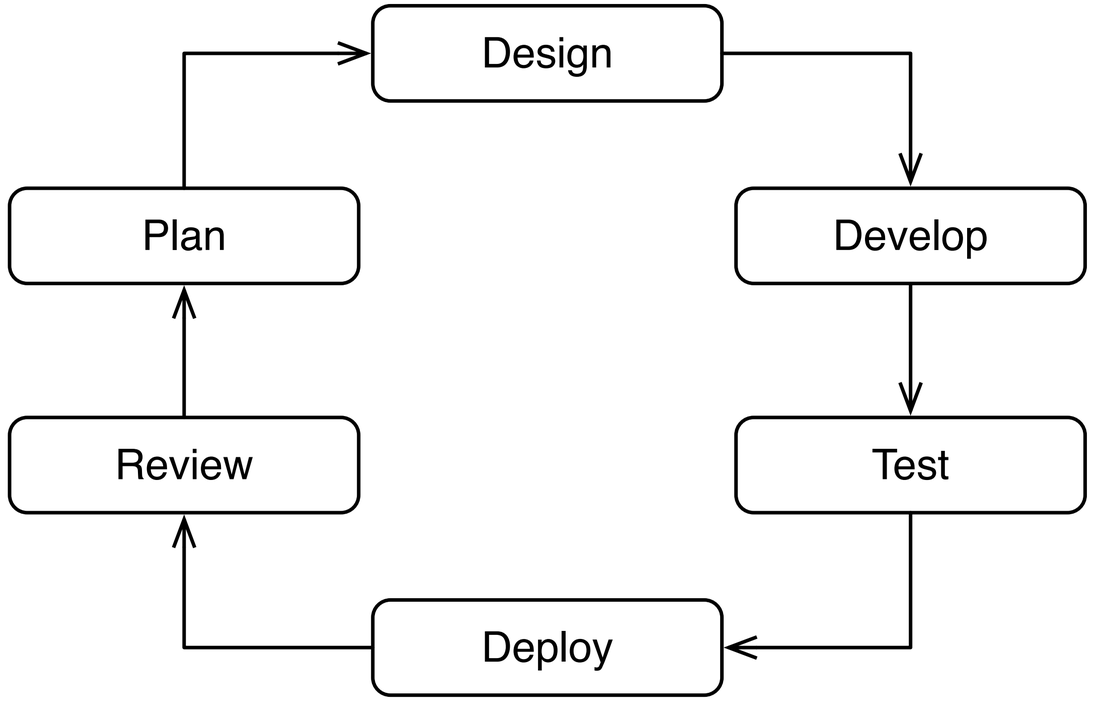
Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

agilemanifesto.org/

Background story: https://agilemanifesto.org/history.html

Agile methodology

- Set of principles and practices for software development
- Designed to accommodate change
- Require a collaborative approach among cross-functional teams
- Examples:
  - Scrum
  - Kanban
  - XP
  - Lean software development
  - Feature-driven development



Agile principles

- **Continuously deliver** value to customers with early and frequent software releases
- **Collaborate daily** across all team members, blending business and technical expertise
- Foster a supportive environment for self-motivated teams, emphasizing direct communication
- Focus on **delivering functional software** as a key progress indicator
- Ensure the **development pace is sustainable** for long-term project health
- Pursue technical excellence, good design, and simplicity
- Engage in **regular self-assessment** to improve effectiveness and processes

Key characteristics

- **Adapt to changing requirements**, even late in a project's development cycle
- Agile projects are broken down into **small, manageable increments**
  - Each increment goes through multiple steps: planning, design, coding, and testing
  - An increments is usually referred to as a "**sprint**" (or an "iteration")
- **Stakeholder involvement is important**: Regular feedback from customers or end-users is integrated into the development process
- **Self-organizing and cross-functional teams**
- **Continuous improvement** of the product, the process, and the team
- **Frequent releases** of the product

Retrospectives

Summary: Need for process control in software development

University of Pittsburgh

Software crisis (~1960s-1980s*)

- Cost overruns, delays, and unreliable software due to unstructured development

Need for Software Development Lifecycle Models:

- Process control ensures quality, predictability, and scalability in software projects
- Ad-hoc development: Code & fix model
  - Lack of planning, maintenance, and scalability
  - Suitable for small, well-understood projects
- Waterfall Model: Sequential, rigid, suitable for stable requirements but not adaptable to changes

- Incremental (add-on), iterative (re-do) vs adaptive (adjust for change) development approaches
- Agile Manifesto: Focuses on collaboration, flexibility, and delivering incremental value

Optional Readings

- CS History
  - https://www.computerhistory.org/timeline/computers/
- ENIAC
  - https://www.computerhistory.org/revolution/birth-of-the-computer/4/78
  - https://spectrum.ieee.org/the-women-behind-eniac
- Agile Manifesto
  - https://agilemanifesto.org/
  - https://www.agilealliance.org/agile101/the-agile-manifesto/

References

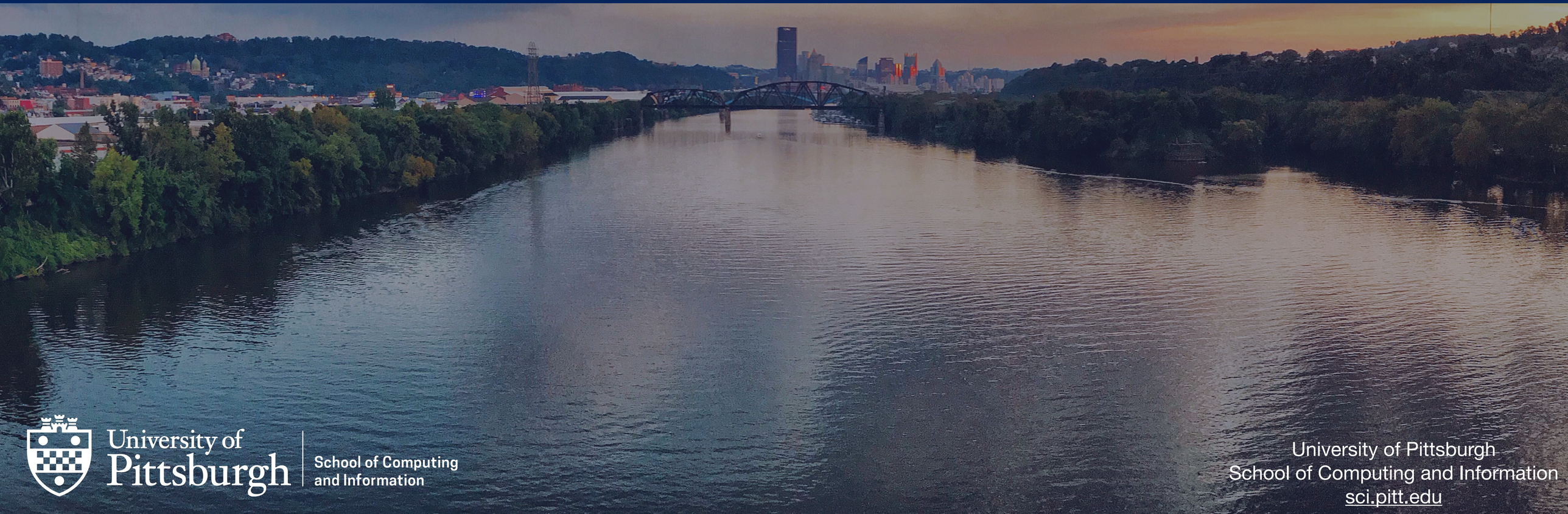
- Bruegge, & Dutoit. Object–oriented software engineering. using UML, patterns, and Java. Pearson, 2009.
- Object Management Group. Unified Modeling Language. Version 2.5.1, 2017
- Agile Alliance, https://www.agilealliance.org/agile101/the-agile-manifesto/

Fall 2025

L02 Introduction to SE

CS 1530 Software Engineering

Nadine von Frankenberg



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