

# Coen 194 Meets This Friday

3:30 pm – 4:35 pm

Bannan 135

# Reflections on Labs So Far

# Help the Professor

# What's This Course About?

- Scaling a cottage industry
- Improving predictability
- Solving the right problems
- Enhancing reliability and maintainability
- Determining and documenting tradeoffs
- Quantifying and incessant improvement

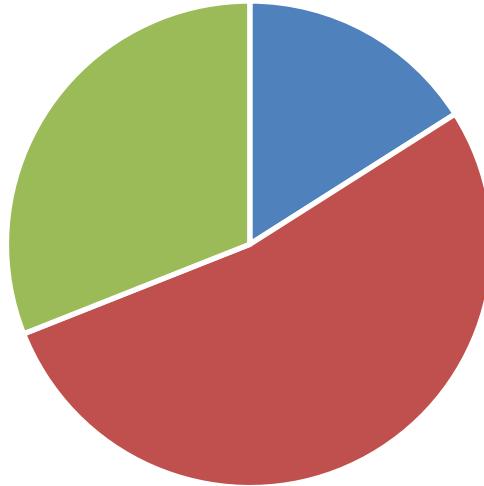
# Why Learn Software Engineering?

- SE is **hard!**
- SE is **expensive!**
- Software is **ubiquitous!**

# SE is Hard!

- Many software systems are large
  - F-22, 2M LOC
  - B-787, 8M LOC
  - High-end car, 100M – 150M LOC
- War stories
  - Fred Brooks, *The Mythical Man-Month*, Addison-Wesley, 1975
  - H Lin, *Scientific American*, v. 253, no. 6, December 1985
- Surveys indicate poor success rates
  - Only 20% of projects are fully successful
  - Approximately 30% of projects are cancelled before completion

# SE is Hard – Johnson 1995



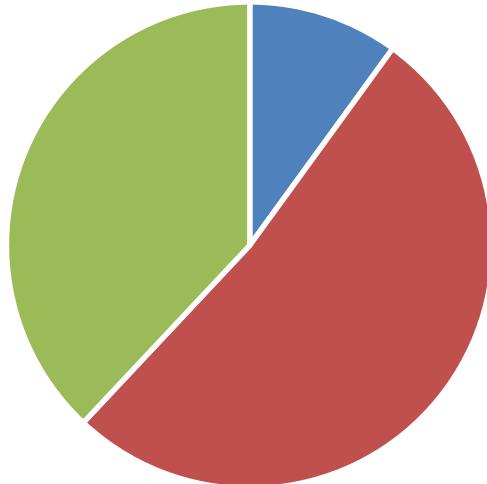
- On Time, on budget ■ Late, over budget, partial
- Terminated or abandoned ■

16% on time, 53% late/over budget/missing functionality, 32% abandoned

Average budget overrun 290%

Average length 320% of original estimate

# SE is Hard – Johnson 2013



- On Time, on budget ■ Late, over budget, partial
- Terminated or abandoned ■

10% on time, 52% late/over budget/missing functionality, 38% abandoned

Data for only large (> \$10 M) projects

# Software is Expensive

- FBI Virtual Case File
  - <https://spectrum.ieee.org/computing/software/who-killed-the-virtual-case-file>
  - \$170M and over four years effort
  - Abysmal performance by software contractor
  - Abysmal oversight by FBI
  - Problems
    - Poorly defined requirements
    - Overly ambitious schedules
    - Inadequate planning for HW acquisition, network deployment, and SW development
- 1994 – FAA advanced automation system - \$2.6B
- 1997 – IRS Tax modernization system - \$4B
- 2004 – Ford purchasing system - \$400M

# Software is Ubiquitous

- Software is in everything
  - Large software systems
  - Almost every interesting thing
    - Planes, cars, phones, refrigerators, thermostats, smoke alarms
- Which explodes the impact of software failures
  - IEEE Spectrum web page of software problems 2005 – 2015
    - <https://spectrum.ieee.org/static/the-staggering-impact-of-it-systems-gone-wrong>

# Why do SE Projects Fail?

- Unrealistic project goals
- Badly defined system requirements
- Poor communication among customers, developers, and users
- Poor project management
  - Inadequate resources
  - Poor status reporting
  - Unmanaged risks
- Bad development practices

# SE Code of Ethics

- [ACM Website on SE Ethics](#)
- PUBLIC – SEs shall act consistently with the public good
- CLIENT AND EMPLOYER – SEs shall act in a manner that is in the best interests of their client and employer consistent with the public interest
- PRODUCT – SEs shall ensure that their products and related modifications meet the highest professional standards possible

# SE Code of Ethics (cont.)

- JUDGMENT – SEs shall maintain integrity and independence in their professional judgment
- MANAGEMENT – SE managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance
- PROFESSION – SEs shall advance the integrity and reputation of the profession consistent with the public interest

# SE Code of Ethics

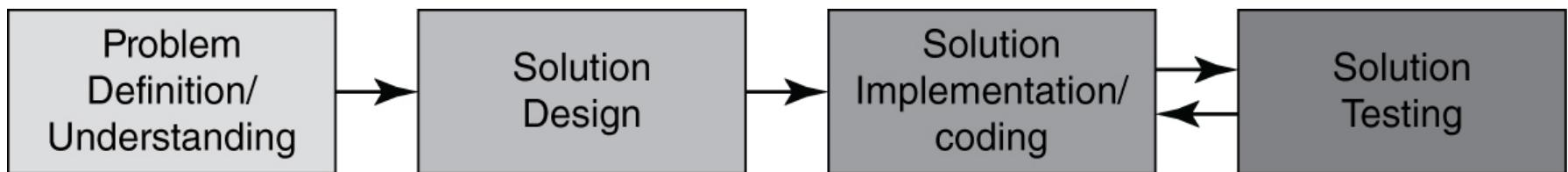
- COLLEAGUES – SEs shall be fair to and supportive of their colleagues
- SELF – SEs shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession

# Acknowledgement

- Some of the materials in these slides are provided courtesy of the textbook authors and publisher

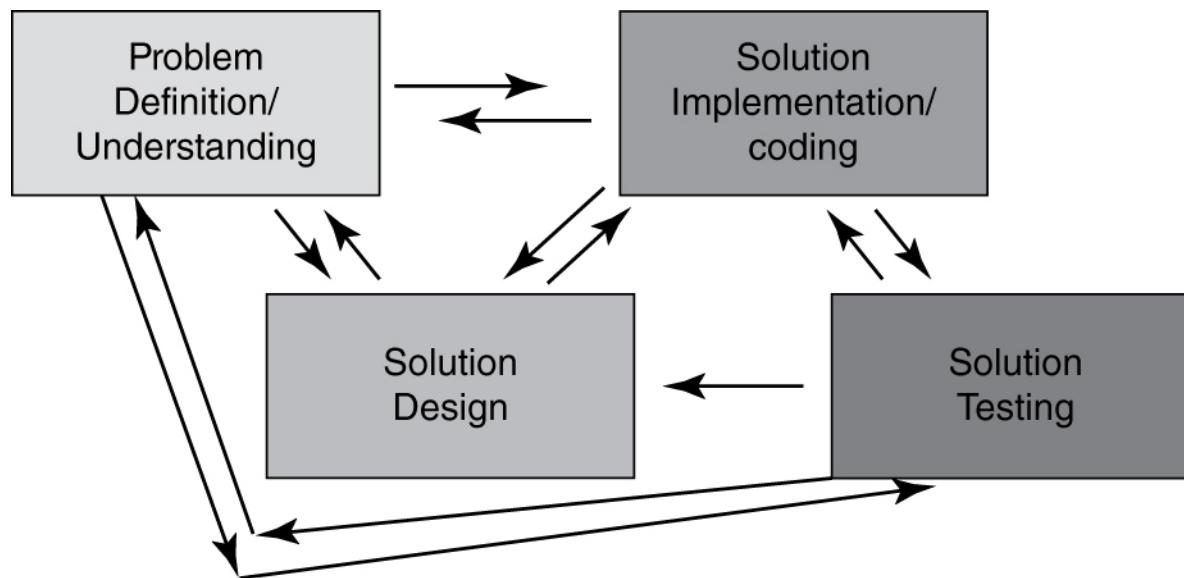
# Caveat

- You'll see a lot of nice neat diagrams



# Caveat (cont.)

- Those are idealized and the reality is much uglier



# Software Engineering

COEN 174

Ron Danielson

Software Engineering Phases

Chapter 1

# Objectives

- Awareness that SE **is** engineering
- Understand the four major phases of SE

# SE Phases

- Requirements determination
  - Define the problem and specify what you are going to build
- Design decisions
  - Define the structure of the solution
- Implementation
  - Build the solution based on the design and established principles
- Testing
  - Verify the solution was built correctly
- Support and maintenance

# Relative Interest

- Survey of research articles on automotive SE
  - Requirements = 35
  - Design = 131
  - Implementation = 248
  - Testing = 246
  - Maintenance = 18

Haghaghkhah, et. al., “Improving the State of Automotive Software Engineering,” *IEEE Computing Edge*, May 2018

# Requirements

- Define and qualify what the system must do
- Typically defined by the client
- Must be prioritized
  - Are negotiable
  - Some may not be implemented at all

# Requirements (cont.)

- **Functional requirements** must be done
  - “The system will...”
- Usually boolean in nature
  - The system either does, or doesn’t, meet the requirement
  - Examples: operations performed, input formats, special cases, error handling procedures

# Requirements (cont.)

- **Non-functional requirements** describe the manner in which the functional requirements must be achieved
  - “The system will be...”
- Usually have a scale associated with them from partial to complete satisfaction
  - Examples: performance, modifiability, usability, configurability, reliability, availability, security, scalability
  - Performance and reliability usually have a quantitative nature

# Requirements (cont.)

- **Design constraints** limit the ways in which the system may operate
  - Constrain the solution, not the problem
  - Examples: platforms, schedule limitations, interface needs, data magnitudes

# Design Decisions

- Are made by the engineer
- Examples: implementation language, framework, hardware (although not always), software architecture

# Implementation

- Following the overall design previously determined, write executable code that performs functions so the system satisfies the requirements
  - Many approaches to process, depending on
    - Number of developers involved
    - Degree of client engagement
    - Tradeoffs agreed on with clients
    - Magnitude of development project

# Testing

- **Validation testing** helps clarify what the system needs (or can) do
  - Helps both engineers and clients understand what will be built, and validates that the system built **will be** correct
- **Verification testing** helps determine whether the system built **is** correct
  - **Unit testing** on small modules and aggregates
  - **Acceptance testing** done by client before they accept the system

# Testing (cont.)

- **White box** testing
  - Allows knowledge of the code
  - Done by a developer
- **Black box** testing
  - No knowledge of code
  - Based on inputs and expected results from those inputs (as defined by the requirements)

# Support and Maintenance

- Support
  - Answering questions, technical help
  - Tracking and fixing bugs
- Maintenance
  - Corrective: fix errors
  - Adaptive: accommodate new platforms
  - Perfective: add new features
  - Preventative: quality improvement for next round of development

# Software Engineering

COEN 174

Ron Danielson

Systems

Chapter 2

# Objectives

- Understand size and complexity issues of systems
- Know how to develop and support a system
- Appreciate the coordination needs of system development and support

# Programs vs. Systems

- More **complexity** in systems
  - Due to **breadth** (number or variety of components)
  - Due to **depth** (complexity and relationships among components)

# Breadth

- Functionality
- Features within each function
- Interfaces
  - Internal and external
- Number and complexity of data types and data structures, as well as volume of data
- Users (and variety of users)
- Example: international language support

# Depth

- Linkages and connections
- More
  - Frameworks
  - Libraries
  - Layers
  - Data sharing across functionality
  - Control passing between functionality
- Can lead to more reuse and reusability, but also greater complexity

# Software [Development] Processes

- A **SD process** is a framework or methodology for implementing a system
- Process specifies
  - A set of tasks to perform
  - The inputs and outputs of those tasks
  - Pre- and post-conditions for each of the tasks

# Technical Considerations

- Simplification
  - Decomposition of problem and solution
  - Modularization
  - Separation of concerns of problem and solution
  - Incremental interations
- **Loose coupling, high cohesion**

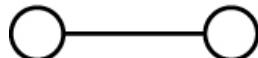
# Technical Considerations (cont.)

- Technology and tools
  - Platform
  - Programming language
  - Database
  - Network infrastructure
  - Configuration management system
  - Version control and build

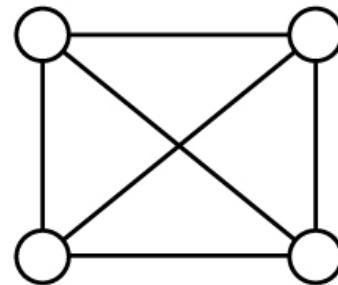
# Non-technical Considerations

- Effort estimation becomes much more important
- Resource allocation becomes essential
  - And skill distribution among implementers
- Coordination of people and resources
- Support and training

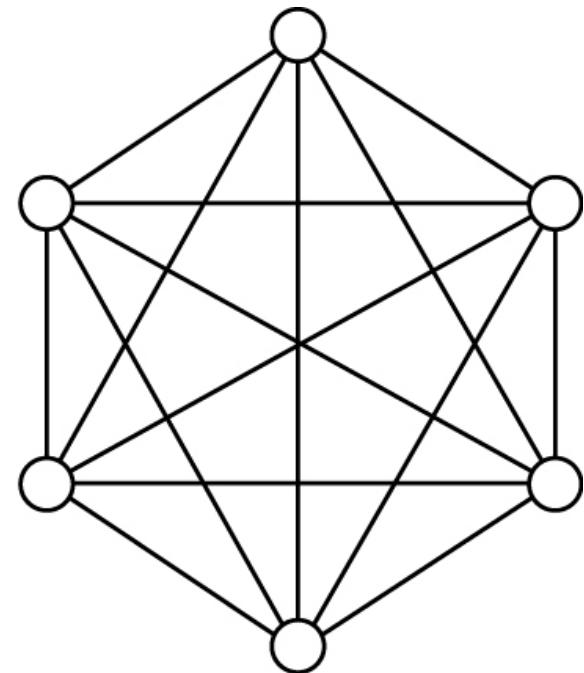
# Non-technical Considerations (cont.)



2 people:  
1 path



4 people:  
possibly 6 paths



6 people:  
increase to  
potentially 15 paths

# Non-technical Considerations (cont.)

- Recommendation for project – **designate leads**
  - Client
  - Design
  - Implementation or coding
  - Test
  - documentation