CIS 422/522
Course Overview

Admin: Projects and Teams
Schedule
Grading

Lecture/Disc: What is Software Engineering?

Contact Information

- Instructor contact
  Stuart Faulk
  faulk@cs.uoregon.edu
  346-1350

  Computer and Information Science
  University of Oregon
  Eugene, OR 97403
- Office Hours: Deschutes 354, after class, by appointment, or any time my door is open
  - I respond most quickly to email

Instructor Background

- Real World Experience (20+ years)
  - R&D U.S. Naval Research Lab
  - R&D Aerospace industry
  - Consulting (various)
- Teaching industry professionals (15+ years)
  - Developed and taught in Oregon Master of Software Engineering
- Emphasis on Software Engineering as an applied discipline (i.e., what actually works)

CIS 422 Course Format

- Single Quarter Project Course
  - Lectures, reading: theory, principles, and methods
  - Projects: learn how to apply SE concepts by doing
  - Project Meetings: learn effective teamwork
  - Project Reviews and Presentations: critique and guidance
- Two project iterations
  - First for perspective on SE issues, team development
  - Second to demonstrate ability to apply lessons learned
- Two exams (midterm, final) assess individual understanding

Emphasis is on Life-Cycle Management and Teamwork

- Participate in collaborative design
- Work as a member of a project team, assuming various roles
- Create and follow project and test plans
- Create the full range of work products associated with a software product
- Complete project deliverables on time
- *Key point: coding is only part of the work*

Projects

- 2 projects: 4 weeks, 6 weeks
  - Project 1: same basic requirements for everyone
    - Simple but extensible application
    - Focus on project planning and teamwork
  - Project 2: a selection of projects
    - Choose among suggestions
    - Propose custom project
- Technically simple, but high expectations
  - Solid freeware quality
  - Complete product includes internal and external documentation, tests

Teams

- Form teams of 4-5 people
  - Project 1: Instructor chooses teams
  - Project 2: May re-form teams
- Project grades are a combination of group grade, individual contributions, and peer evaluation
  - Overall grade for project
  - Evaluate individual contributions (logs)
  - Group Member Evaluation (GME) by teammates may significantly raise or lower grade

Grading

- 60% Projects (20+40)
  - Includes presentations, intermediate deliverables
- 30% Exams (15+15)
  - Test for understanding of lectures & reading
- 10% Class Participation: includes but is not limited to...
  - Attendance at class, team meetings
  - Contributing the discussions, class exercises
  - Appropriate behavior in the classroom (i.e. no cell phones, beepers, trolling web)
- Questions?

What is Software Engineering?

The "Software Crisis"

- Have been in "crisis" since the advent of "big" software (roughly 1965)
- What we want for software development
  - Low risk, predictability (time, cost, functionality, quality)
  - Lower costs and proportionate costs
  - Faster turnaround
- What we have:
  - High risk, high failure rate
  - Poor delivered quality
  - Unpredictable schedule, cost, effort
- Characterized by **lack of *control*** (inability plan the work, work the plan)

Symptoms of the "Crisis"

- One of every four large software project is cancelled
- Average project overshoots schedule by 50%, large project often do much worse
- 75% of large systems are do not operate as intended
  - E.g., Ariane 5, Therac 25, Mars Lander, DFW Airport, FAA ATC, Universal Credit, etc., etc.
  - Many fail to deliver a single working line of code
- Really the "state of practice"

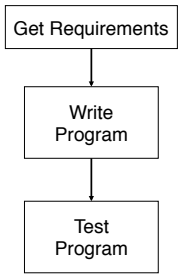
Discussion Context

- Focus large, complex systems
  - Multi-person: many developers, many stakeholders
  - Multi-version: intentional and unintentional evolution
- *Quantitatively* distinct from small developments
  - Software complexity grows non-linearly with size
  - Communication complexity grows exponentially
- *Qualitatively* distinct from small developments
  - Multi-person implies need for organizational functions (management, accounting,), policies, oversight, etc.
  - More stakeholders and more kinds of stakeholders
- Rule of thumb: project starts to be "large" development team can't fit around a table.

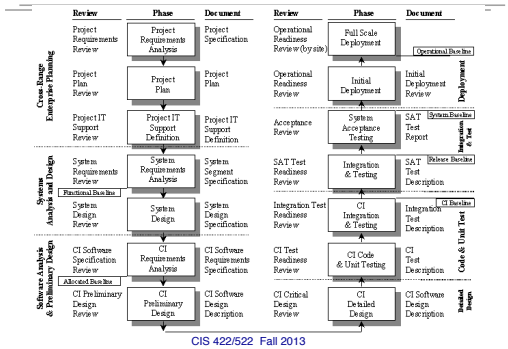
Implications

- Small system development is driven by technical issues (I.e., programming, technical understanding)
- Large system development is dominated by organizational issues
  - Managing complexity, communication, coordination, etc.
  - Projects fail when these issues are inadequately addressed
- Key Lesson #1: **programming ≠ software engineering**
  - Techniques that work for small systems fail utterly when scaled up
  - Programming skills alone won't get you through real developments (or even this course)

Programming View



DoD Software Life Cycle



Origins of SE

- Term "software engineering" was coined at 1968 NATO conference:
  - "Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."
- Response to "software crisis"
  - Failed developments
  - Lack of critical qualities (e.g., performance, safety, reliability, maintainability)
  - Budget and schedule overruns
- Desire for software development to be more like other engineering disciplines
  - Analytical, predictable, manageable
  - But, stated as an aspiration, not the state of practice

What has changed since '68?

- Incorrect to conclude that no progress has been made
  - Better understanding of issues
  - Substantial improvements in programming languages, tools
  - Better understanding and control of software processes
- But the problems have also changed
  - Large developments now are orders of magnitude more code than in 1968
  - Improved capabilities often overcome by larger problems, greater complexity

What hasn't changed?

- Still not an engineering discipline in classic sense
  - Lack of applied mathematics and systematic methods to develop and assess product properties
  - Not taught, licensed, regulated, or recognized as an engineering discipline
- But practitioners often don't apply what we know
  - Existing methods, models often not understood or used in industry
  - Little attention is given to processes or products other than code
  - Quality of products depends on *qualities of the individuals rather than qualities of engineering practices*
- Development continues to be characterized by **lack of *control***

Historical Comparison

Pre-Industrial

The Craftsman

Post-Industrial

The Factory

Which best characterizes software?

Pre-Industrial

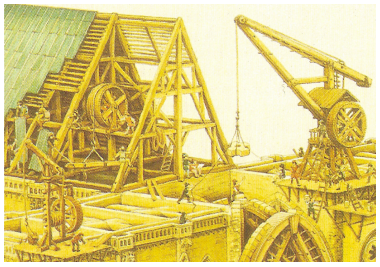
- Craftsman builds product
  - Builds one product at a time
  - Each product is unique, parts are not interchangeable
  - Quality depends on craftsman's skill – product of training, experience
  - Many opportunities for error
- Focus on individual products
  - Customization is easy
- Scaling is difficult
  - Parts are not interchangeable
  - No economy of scale
  - **Control problems rise *exponentially* with product size!**

Post-Industrial

- Products produced by machines
  - Quality depends on machines & manufacturing process
  - Production requires little training or experience
- Focus on developing the **means of production**
  - Craftsman builds means to build product (tools, factory)
  - Customization is difficult
- Easily scales
  - Parts are interchangeable
  - Products are alike
  - Economies of scale apply

The "Big Project" Problem

- **Control problems rise *exponentially* with product size!**
- **Each interface must be hand fit**



View of SE in this Course

- The ***purpose of software engineering*** is to *gain* and *maintain* intellectual and managerial control over the products and processes of software development.
  - "Intellectual control" means that we are able make rational choices based on an understanding of the downstream effects of those choices (e.g., on system properties).
  - Managerial control similarly means we are able to make rational choices about development *resources* (budget, schedule, personnel).
- Memorize this!

Control is the Goal

- Both are necessary for success!
- Intellectual control implies
  - We understand what we are trying to achieve
  - Can distinguish good choices from bad
  - We can reliably and predictably build to our goals
    - Functional behavior
    - Software Qualities (reliability, security, usability, etc.)
- Managerial control implies
  - We make accurate estimations
  - We deliver on schedule and within budget
- Assertion: managerial control is not really possible without intellectual control (no matter what the Harvard School of Business says)

Course Approach

- Will learn practical methods for acquiring and maintaining control of software projects
- Intellectual control
  - Methods for software requirements, architecture, design, test
  - Modeling methods and notations
- Managerial control
  - Planning and controlling development
  - Process models addressing development issues (e.g. risk, time to market)
  - People management and team organization
- Caveat: we can only simulate the problems of large developments

Questions?

Assignment

- Fill out and return the team member survey
- Review web site (syllabus, etc.)
  - Read the project description
  - Do readings specified in the schedule

Questionnaire

- Purpose
  - Formation of balanced project 1 teams
  - Beginnings of grade database
- Fill in
  - Name (family, given), What you would like to be called
  - Proficiencies
    - 1 low, 3 average, 5 high
    - 5 means you have extensive experience, can apply the skill immediately with good results
    - 3 means you have used the technology, may need some review