CIS 422/522
Course Overview

Admin: Projects and Schedule
Grading
Lecture/Disc: What is Software Engineering?

Contact Information

• Instructor contact
  Stuart Faulk
  faulk@cs.uoregon.edu
  346-1350
  Deschutes 354
  Computer and Information Science
  University of Oregon
  Eugene, OR 97403

• Office Hours: 10:30-11:30 class days, 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 (DoD, Sharp, Sun, etc.)
- Teaching industry professionals (15+ years)
  - Oregon Master of Software Engineering
- Perspective 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 evaluations: critique and guidance
- Two project iterations
  - First for perspective on SE issues, team development
  - Second to demonstrate ability to apply lessons learned
- Two exams assess individual understanding
  (midterm, 2nd midterm)
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 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
    - Understand what can go wrong
  - Project 2: a selection of projects
    - Instructor suggested or team choice
    - Focus on disciplined development
- Technically simple, but high expectations
  - Solid freeware quality application
  - Complete documentation: requirements, design, test, user guides
Teams

- Form teams of ~5 people from surveys
  - At least one common programming language
  - Cross-section of skills
- Project grades are a combination of group grade, individual contributions, and peer evaluation
  - Overall grade for project
  - Evaluation of individual contributions
    - Peer evaluation by teammates
    - Record of contributions from Developer Log

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
  - Participation in class discussions, interactive questions
  - Appropriate behavior in the classroom (i.e. no cell phones, beepers, trolling web)
Grading Constraints

To pass the course you must meet all of these criteria:

• 65 or better on the project
• 65 or better average on the exams
• Appropriate team interactions (i.e., appropriate language, civil, professional, etc.)

Class Website

• Use class website to track class events
• Schedule page most important
  – Lecture schedule, link to slides
  – Readings due for each lecture
  – Project due dates
  – Examples of work products
• Home page: announcements
• Project page: project description, constraints
• Project grading: how work will be evaluated
Additional Resources

- Assembla: team online collaboration sites
- Piazza: forum for discussion, questions (including anonymous)
- Provide summaries of lectures
- Video lectures: in place of in-class lectures for some classes; links provided as needed

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
  - Inconsistent 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 do not operate as intended
  - E.g., Ariane 5, Therac 25, Mars Lander, FAA ATC, Universal Credit, Cover Oregon, etc.
  - Many fail to deliver a single working line of code
- Really the “state of practice”
Discussion Context

- Focus on 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
  - Problem understanding, 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

1. Get Requirements
2. Write Program
3. Test Program

DoD Software Life Cycle

- Review: Project Requirements Review
- Plan: Project Plan
- Document: Project Plan
- Review: Operational Readiness Review
- Plan: Operational Readiness Plan
- Document: Operational Readiness Plan
- Review: Initial Deployment Review
- Plan: Initial Deployment Plan
- Document: Initial Deployment Plan
- Review: System Acceptance Review
- Plan: System Acceptance Plan
- Document: System Acceptance Plan
- Review: SAT Test Review
- Plan: SAT Test Plan
- Document: SAT Test Plan
- Review: Integration & Testing
- Plan: Integration & Testing Plan
- Document: Integration & Testing Plan
- Review: CI Test & Unit Testing
- Plan: CI Test & Unit Testing Plan
- Document: CI Test & Unit Testing Plan
- Review: CI Detailed Design
- Plan: CI Detailed Design Plan
- Document: CI Detailed Design Plan
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”

• Desire for software development to be more like mature 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
  – Improved capabilities often overcome by larger problems, greater complexity
  – Orders of magnitude more code, faster pace of technology, accelerated delivery schedules, etc.
What has not 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, or regulated as an engineering discipline (most of USA)
- Worse, practitioners often don’t apply what we know
  - Existing SE methods, models often not understood or used in industry
  - Little attention is given to processes or products other than code
  - Upshot: quality of products depends on qualities of the individuals rather than qualities of engineering practices
- Development continues to be characterized by **lack of control**

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!
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
  – What to produce, how to make decisions, how to check correctness

• Managerial control
  – Planning and controlling development
  – Process models addressing development
  – People management and team organization

• Caveat: we can only simulate the problems of large developments
Assignments

• Read through the class web site
  – Make sure you understand what is expected of you and how the course is graded
  – Understand how the schedule page works, this should be checked before class
• Read the project description
• Read through the Team Roles page and consider which roles interest you
• Read the Process Models reference before next class

Questions?