CIS 422/522 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
120 Deschutes Hall
University of Oregon
Eugene, OR 97403

- Office Hours: 2:00 – 3:00 after class or by appointment

Instructor Background

- Real World Experience
  - R&D U.S. Naval Research Lab (15 years)
    - Embedded and real-time systems
    - Software Engineering methodology (requirements, design, concurrency)
  - R&D Aerospace industry (5 years)
    - Requirements methods, software product lines
  - Consulting (various)
- Academic (15 years)
  - Developed and teach in Oregon Master of Software Engineering (industry professionals)
  - Research in software engineering
- Potential weaknesses
  - Do not use many current programming technologies (so I cannot help with technology)

CIS 422 Course Format

- Single Quarter Project Course
  - Lectures: Foundations and background
  - Projects: Learn how to apply SE concepts
  - Project Meetings: Learn teamwork
  - Project Reviews and Presentations: Critique and guidance
- Two team projects
  - First for perspective on SE issues
  - Second to demonstrate learning and ability
- Two exams (midterm, final) address 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 a project and test plan
- Create the full range of documents associated with a software product
- Complete a project on time
- *Key point: the focus is not on coding!*

Projects

- 2 projects: 4 weeks, 5 weeks
  - Project 1: Small selection
    - Same basic requirements for everyone
  - Project 2: TBD
    - You will propose projects
- 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: Choose your own teams
    - the most important decision you will make
- Project grades are group grades
  - Every member responsible for every part
  - Members will evaluate each other (GMEs)

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

Weekly Schedule

- Tuesday and Thursday lectures
  - Mix of lectures, discussions, group exercises
  - Some lecture times or parts thereof will be used for team meetings and project discussions
- Meetings with the professor
  - Design reviews
  - Grading

Grading

- 55% Projects (20+30)
  - Includes presentations, intermediate deliverables
  - Weighted toward *non-code products*
- 35% Exams (15+20)
  - Two midterms; no final exam
- 10% Class Participation
  - Includes but is not limited to...
    - Attendance (required)
    - Contributing the discussions (can also be done via email)
    - Appropriate behavior in the classroom (i.e. no cell phones or beepers)

What is Software Engineering about?

The “Software Crisis”

- Have been in “crisis” since the advent of “big” software (roughly 1965)
- What we want for software development
  - Low risk, predictability
  - 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 projects overshoot schedule by 50%, large project do much worse
- 75% of large systems are failures in the sense that they do not operate as intended
- 60% of them fail to deliver a single working line of code
- E.g., Ariane 5, Therac 25, Mars Lander, DFW Airport, FAA ATC etc., etc. (See examples in Text)

Discussion Context

- Focus large, complex systems
  - Multi-person: many developers, many stakeholders
  - Multi-version: intentional and unintentional evolution
- Quantitatively distinct from small developments
  - Complexity of software (e.g. rises non-linearly with size)
  - Complexity of communication rises exponentially
- Qualitatively distinct from small developments
  - Multi-person introduces need for organizational functions (management, accounting, marketing), policies, oversight, etc.
  - More stakeholders and more kinds of stakeholders
- Rule of thumb: project starts to be "large" when group developing a single product can't fit around a table.

Software is Pre-Industrial

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!**

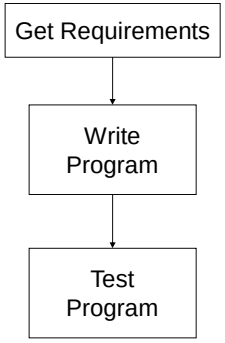
Post-Industrial

- 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

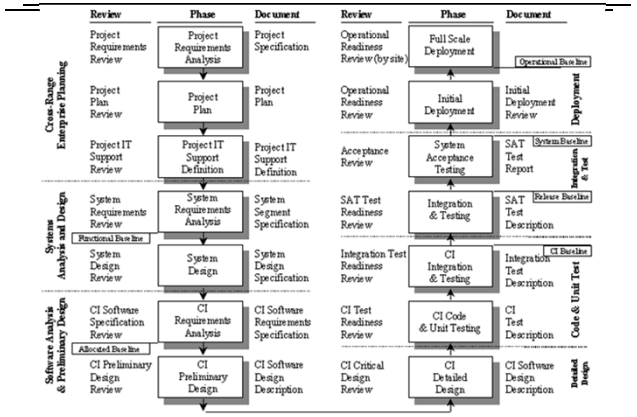
Implications

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

Programming View



DoD 2167 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” manifest by systems that
  - Failed to provide desired customer functionality
  - Lacked critical qualities (e.g., performance, safety, reliability)
  - Overran budget and schedule (hugely)
  - Were difficult to change or maintain
- Desire for SE to be more like other engineering disciplines
  - Analytical, predictable, manageable
  - State as an aspiration, not statement of existing condition

Has anything changed?

- Incorrect to conclude that no progress has been made
  - Substantial improvements in programming languages, tool
  - Better understanding and control of processes
- But the problems have also changed
  - Large developments now are orders of magnitude more code than in 1968
  - Improved capabilities are overcome by larger problems, greater complexity
- Note: “software crisis” is a euphemism for “state of the practice”

What hasn't changed?

- Still not an engineering discipline in classic sense
  - Implies use of applied mathematics and systematic methods to develop and assess product properties
  - These tools are immature where they exist at all
  - Software "engineering" is not taught, licensed, regulated, ore recognized as an engineering discipline (e.g., by engineers)
- But we often don't apply what we know
  - Existing methods, models often not understood or used in industry
  - Little attention is given to process 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*** (inability plan the work, work the plan)

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).

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 methods for acquiring and maintaining control of software projects
- Intellectual control
  - Methods for software requirements, architecture, design, test
  - Notations, verification & validation
- 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 really only scratch the surface in 10 weeks (but it's important)

Assignment

- Reading:
  - Text: Chapters 1, 2, and 3
  - 522 Read: “A Rational Design Process: How and Why to Fake it,” Parnas & Clements
- Review web site (syllabus, etc)
- Project: prepare for first project meeting (team assignments)
  - Read project description
  - Think about what role you want to play