CS121 - Software Development

- Administrative Introduction
  - course goals and focus
  - course structure and grading
- Introduction to Software Development
  - why is Software so difficult?
  - what are the hard problems?
  - elements development methodology

Instructor: Mark Kampe

- not regular faculty, not an academic
- I am a retired engineer
  - spent 30 years building operating systems
  - done every job in an engineering organization
  - worked for small and large companies
  - active in architecture, process, and training
- I teach for fun
  - and to ensure you learn some engineering

Contact Information

- Office: Edmunds 2-221
  extension 70969
- Office Hours: Tu/Th 10-12,12:45-2:15
  and by appointment
- E-mail: markk@cs.pomona.edu

Why this course is important

- All of you will develop complex software
  - whether or not you work as programmers
- Software Construction is difficult
  - even simple programs can be hard to build
  - even carefully written programs behave badly
- This is the study of
  - problems inherent in software construction
  - tools and processes for addressing them
  - ways to improve your chances of success

Why Projects Fail

Rank	Challenged	Failure
1	Lack of user input	Incomplete requirements
2	Incomplete requirements/specs	Lack of user involvement
3	Changing requirements/specs	Lack of resources
4	Lack of executive support	Unrealistic expectations
5	Technological incompetence	Lack of executive support
6	Lack of resources	Changing requirements/specs
7	Unrealistic expectations	Lack of Planning
8	Unclear objectives	Didn't need it any longer
9	Unrealistic time frames	Lack of IT management
10	New technology	Technological illiteracy

Requirements 30% | Planning 25% | Management 20% | Technological 7%

Focus of this course

- medium-sized projects
  - multi-staff-month to multi-staff-century
- broad coverage
  - overview the full range of development activities
- comparative approach
  - not teaching one particular development discipline
  - an exploration of issues and approaches
- practical focus
  - what you'll use, rather than what's been written
- driven by a set of "key learning objectives"

Key Concepts and Issues

- I am here to help you understand these
  - They are fundamental to software construction
  - I believe they will be of great value to you
- The lectures are built around them
  - we will discuss them and their applications
- The tests are built around them
  - can you describe or discuss them?
  - can you apply them to problem situations?
  - can you use them to gain new insights?

Representations and Techniques

- I am here to introduce basic methodology
  - languages you will understand and speak
  - skills you will develop and apply
  - processes you will understand and follow
- We will read about and discuss them
  - some will be demonstrated in class
- The Labs are designed around these
  - you will use all of these skills & techniques

Texts and Reading

- There will be daily reading assignments
  - they will average about 40 pages per lecture
- McConnell, *Code Complete, 2nd ed*
  GOOD: clear, focused, and practical advice
  WEAK: little discussion of issues and principles
- This will be supplemented by <u>many</u> papers
  - GOOD: most of them short and highly focused
  - WEAK: they are of varying depth & quality

Interactive Lectures

- Lecture periods will not be used to
  - review subjects in the assigned reading
  - you are expected to have done the reading
- Lecture periods will be used to
  - clarify the reading and work examples
  - synthesize conclusions from divergent sources
  - interactively explore implications & applications
- all lecture slides will be posted on-line
  - to simplify your note-taking and study
- participation is a critical part of learning process
  - significant contributions will earn extra credit

Quizzes

- When: first five minutes of lecture period
- Scope: that day's assigned reading
- Format: closed book
  - 4-6 short and simple questions
- Goals:
  - check your familiarity with key concepts
  - force you to do the assigned reading
  - enable you to get more out of the lectures

Exams

- When: seventh and fifteenth weeks
- Scope: last 7 weeks of reading & lectures
- Format: closed book
  - 10-15 short essay questions
  - very straight forward (if you're keeping up)
- Goals
  - test understanding of key concepts
  - test ability to apply principles to real problems

Projects

- One large problem, in three parts
  - requirements, architecture, review
- One small problem, in five parts
  - reverse engineering, specifications,
  - detailed design, design review, test plan
- Goals:
  - develop practical software development skills
  - develop teamwork skills
  - these are not programming projects

Course Web Site

- http://cs.pomona.edu/classes/cs121
  - reading, lecture, quiz and exam schedule
  - supplementary reading materials
  - project assignments, information, materials
  - copies of all lecture slides
  - solutions to exam/homework problems
- Sakai
  - grades, announcements, discussions

Course Grading

- Final course grade is a function of:
  - concepts 45% (measured on 2 exams)
  - skills 45% (exercised in 8 projects)
  - preparation 10% (daily quizzes)
  - with bonus points for discussion participation (quality, not quantity)
- I look at curve to assign final grades
  - but expect it to be close to 90/80/70/60

Grading - Partial & Extra Credit

- Partial Credit
  - available on all quizzes, exams and projects
  - points are not merely for the final answer
    - points for a clear understanding of the problem
    - points for a reasonable approach to the problem
    - points for correct elements in a flawed solution
- Extra Credit
  - extra credit exam problems
  - for any answer that shows significant insight

Late Assignments & Make-Ups

- Quizzes there are no make-ups
- Homework may not assign, no points for it
- Exams for (documented) medical disability
  - different tests, given after end of the semester
- Projects each person has 3 slip-days (team has sum of individuals') after that it is 10% off per day

Grade Changes

- I will always fix any grading error
  - if I mis-record a score, show me the original
- I will always explain how I score problems
  - I use very detailed grading criteria
- I will always **consider** re-scoring a problem
  - if I miss or mis-understand your answer
  - but only if your answer is better than I thought
- I never change grades due to "hardship"
  - but I do give lectures on personal responsibility

Software v.s. Hardware

- How Software is easier
  - software costs less to build and ship
  - software can be replicated perfectly
  - software does not experience "wear"
- How Software is harder
  - software products are much more complex
  - software is very difficult to inspect or test
  - few engineers/technicians truly understand it

Why Software is so Complex

- intrinsic complexity
  - size & complexity of functional requirements
  - number of "moving parts", modes of interaction
  - number of other systems with which it interacts
  - number of relevant environmental factors
- must meet ever-changing requirements
  - ever-increasing user and data loads
  - ever-evolving interface requirements
  - functional requirements change at web speed

The Legacy Software Problem

- H/W obsolescence is not a major problem
  - most hardware "wears out" long before that point
  - next gen h/w will be spec'ed for next gen needs
- Software does not "wear out" (sic)
  - it will have the opportunity to become obsolete
- Enterprises don't replace mission critical s/w
  - major investments in acquisition and training
  - changes are almost inevitably highly disruptive
- S/W is forced to survive/evolve much longer
  - places greater demands on initial design
  - greatly complicates future maintenance

A Formula for Failure

- today's dominant s/w development paradigm:

```
do {    have an idea;
        build it;
        ship it;
        see how it works;
        } until (we give up);
```

- we aren't sure what we should build
- we don't know when it will be done, at what cost
- we don't know how good it is going to be
- we can't even measure how good it is!

Software Engineering

The establishment and use of sound engineering principles, in order to reliably and economically obtain software that satisfies the user's requirements, is reliable, and works efficiently in real deployment environments.

Basic Methodology

- understand the problem
  - project definition, requirements development
- plan the solution
  - architectural design, modeling, prototyping
  - project planning (risks, resources, schedules)
  - User Interface & component design
- execute the plan
  - reviews, implementation, testing, integration
  - validation, deployment, support

Common S/W Delusions

- We've already got enough policies
- Process takes more time than it saves
- You'll know how long it will be when I finish
- If it's late, we can add more people
- It works for me, ship that sucker
- Once it's done, we can see how good it is
- If it has bugs, we'll find and fix them

Project 1A - Requirements

- Focus – skill development
  - sub-tasks, schedules, status tracking, post mortem
  - project inception and requirements
- ASAP (before next class)
  - form teams, come up with a product concept
  - read the Project 1A description on the web
- over the next four weeks
  - elaborate concept, research existing products, brainstorm initial requirements, gather customer requirements, reconcile conflicts, write final requirements, sell product concept, post-mortem

Reading for next lecture

- McConnell: 28.2
- Spolsky: 12 Steps to Better Code
- Kampe:
  - Reproducibility and Control
  - Surviving Large Projects
  - Post Mortems
- Wingerd: SCM Best Practices

Back up slides

Homework

- When: may be given a few times
- Scope: a few troublesome areas
- Format: a few focused problems
- Goals:
  - reinforce key concepts, issues & skills
  - test ability to apply key concepts
  - work areas where students often have trouble
- Grading: not included in course grade
  - I will correct and return for your feedback

Academic Honesty

- Academic Dishonesty cheats us all
  - devalues grades & degrees,
  - threatens our accreditation
- My Policy
  - zero tolerance, no second chances
  - no warnings, I report everything
- Don't test me

The Rules - Quizzes/Exams

- do not copy anyone else's answers
  - if you study with a friend, sit apart
- no talking or exchanging notes during test
  - if you have a question, raise your hand
- no use of notes or text on any test
  - if you need a dictionary, tell me before the test
- no removal of test materials from classroom
  - all tests must be turned in to the instructor

The Rules - Projects

- You must prepare your own work products
  - in projects you can work with team-mates
  - don't get solutions from other teams
  - don't submit solutions from web or elsewhere
    - if you didn't write it yourself, cite the source
- Protect yourself
  - don't share your solutions with others
- Some projects involve meetings w/others
  - requirements gathering and reviews are OK

The Rules – Homework

- It is OK to study with friends
  - discussing problems helps us understand them
- You must do your homework by yourself
  - do not submit another student's work
  - do not submit solutions you find on the web
  - if you do research **cite your sources**
- I decide when assignments are too similar
  - and forward them immediately to the Dean

Why We Ship Bad Software

- Time is more important than quality
  - our customers need the functionality
  - we need to book the income this quarter
  - we already have a bad "on-time" reputation
- We don't actually know how bad it is
  - our testing methodology inadequate
  - we don't know how it will be used anyway
- Developers are usually goaled on delivery
  - bugs are someone else's problem

The Costs of Bad Software

- Costs to consumers
  - increased costs of ownership
    - lost and reduced productivity/business
    - direct costs of management and support
  - costs of software failures
    - loss of service, revenue, and customers
    - destruction (& death) directly caused by bugs
- Costs to producers
  - cost of late, over-budget, failed projects
  - cost of support greatly exceeds development