Software Engineering

- Administrative Introduction
  - course goals and focus
  - course structure and grading
- Introduction to Software Development
  - why is Software so difficult?
  - what are the hard problems?
  - development methodology
- Project 1 … hit the ground running

Instructor: Mark Kampe

- not regular faculty, not academic
- I am a retired engineer
  - 40 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: McGreggor 312
- Office Hours: T/R 11-12, late in lab
- plus Zoom (incl Sat/Sun for projects)
- E-mail: mark.kampe@gmail.com

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 A	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 A	Technological illiteracy A

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

Focus of this course

- medium-sized projects
  - multi-staff-month to staff-century
- broad coverage
  - overview the full range of development activities
- comparative approach
  - not favoring particular development disciplines
  - exploration of issues and a range of 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 40-50 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

- I would like to minimize using lectures to
  - review subjects well covered in the assigned reading
  - you are expected to have done the reading
- I will attempt to use them to ...
  - clarify and reinforce key results, work examples
  - synthesize conclusions from divergent sources
  - <u>interactively</u> explore implications & applications
- your participation critical to learning process
  - I need you to help drive discussions
- all lecture slides will be posted before lecture
- post-lecture feedback

Lab Sessions

- Focus entirely on projects
  - no lectures, few planned exercises
  - questions about this week's deliverables
  - discuss the path to next week's work
- Get help before Sunday due dates
  - a dependable whole-team time
  - I am available to talk about anything
  - help with problems you are encountering (assumes you have already encountered)

Quizzes

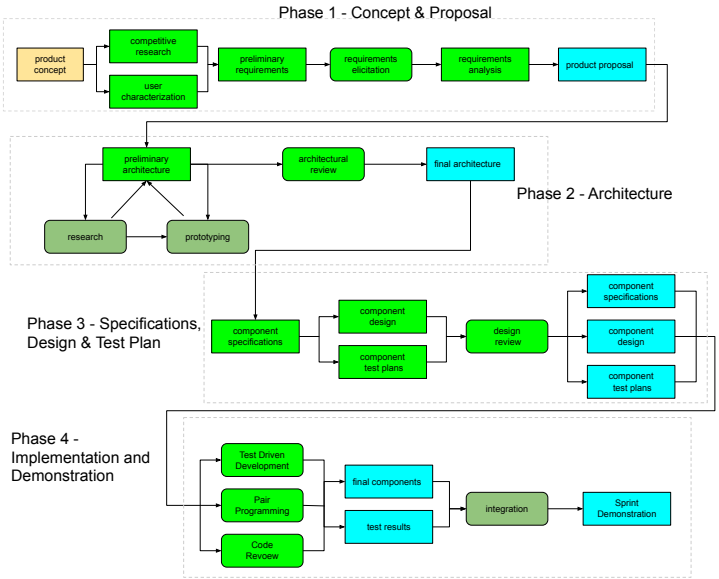
- When: due before start of each lecture
- Scope: that day's assigned reading
- Format: closed book, timed
  - 4-5 short simple questions w/short answers
- Goals:
  - check your familiarity with <u>key concepts</u>
  - force you to do the assigned reading
  - enable you to get more out of the lectures

Exams

- When: mid-course and during finals week
- Scope: reading & lectures
- Format: closed book, timed
  - 10-15 short essay questions
  - straight-forward (if you're keeping up)
- Goals
  - understanding of key concepts
  - ability to apply principles to new problems

Projects

- I … develop product proposals
- II … develop/review architecture
- III … specification, design, planning
- IV … implementation
- Goals:
  - develop practical software development skills
  - develop teamwork skills
  - much more than "programming projects"



Course Web Site(s)

- www.cs.hmc.edu/courses/2024/spring/cs181aa
  - reading, lecture, quiz and exam schedule
  - supplementary reading materials
  - project assignments, information, materials
  - copies of all lecture slides
- Canvas:
  - discussions, help & topic requests
  - quizzes, submissions, feedback, grades
- Zoom - any-time Q&A
- Gradescope - exam feedback

Grutor Support

- Help with Academic Material
  - contribute to Piazza questions
  - scheduled weekly discussion sessions
  - review session in preparation for exams
- Help with Projects
  - discuss project issues and approaches
  - review deliverables before submission
- not involved in exam/project grading

Course Grading

- Final course grade is a function of:
  - preparation 10% (daily quizzes)
  - concepts 40% (measured on 2 exams)
  - skills 50% (exercised in projects)
- I do not grade on a curve
  - expect breaks a little below 90/80/70/60

Grading - Partial & Extra Credit

- Partial Credit
  - – available on all 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
  - – any answer that shows significant insight

Late Assignments & Make-Ups

- Quizzes no make-ups, but can take early
- Exams for (documented) medical disability
  - – different tests, given some time later
- Projects each person has a few slip-days
  one day's deliverables, for one day
  (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 misunderstand your answer
  - – but only if your answer is better than I thought
- I never change grades due to "hardship"

Software vs. Hardware

- How Software is easier
  - – software costs less to build and ship
  - – software can be replicated perfectly
  - – software does not "wear out" (sic)
- 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 less a 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

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 usually goaled on delivery
  - bugs are support’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

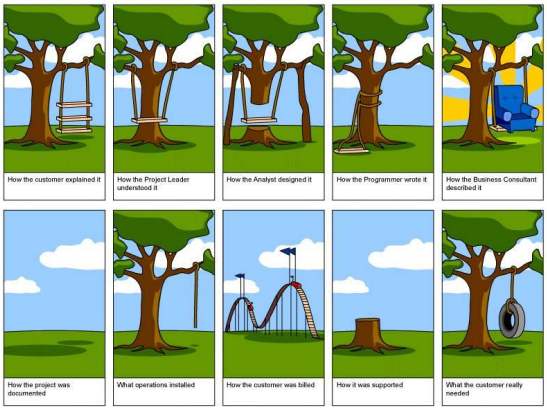
A Formula for Failure

- 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!

Many ways to go wrong



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.

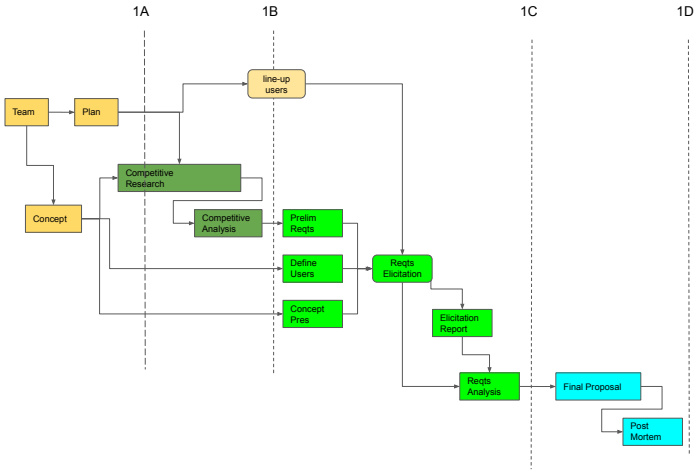
Development Methodology

- 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

Project 1: concept to proposal

- Focus – skill development
  - sub-tasks, schedules, status tracking, post mortem
  - project inception and requirements
- ASAP (by next class)
  - form teams (~3-5), come up with a product concept
  - read the Project 1 description on the web
  - think about what those tasks and deliverables mean
  - be prepared to ask questions, discuss concept
- over the next few weeks
  - elaborate concept, research existing products, brainstorm initial requirements, gather customer requirements, reconcile conflicts, write final requirements, write proposal, post-mortem

Phase 1: Concept to Proposal



For the next lecture

- McConnell: 28.2 Configuration Management
- Spolsky: 12 steps to better code
- Kampe:
  - Reproducibility and Control,
  - Surviving Large Projects,
  - What is a Post-Mortem
- Shaefer: Diablo II Post-Mortem
- Project 1
  - share ideas (Canvas), form tentative teams
  - read and discuss the assignment

Back up slides

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

The Rules - Quizzes/Exams

- do your own work, no help from anyone
  - if you have a question, contact me
- no talking or exchanging notes during test
  - if you have a question, contact me
- no use of notes or on-line resources
- Some will be taken electronically
  - your honor code makes me comfortable w/this

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

Deeper 

- What are "new technology" problems?

  New technologies are, by definition, not mature. Their capabilities may not be well understood, and the tools may be flaky.
- Can "programmers" solve them?

  Not in the short term. The risks must be assessed, through research and prototyping, and plans must be developed to mitigate those risks. These are design, planning, and management problems.

Deeper

- "technological illiteracy" problems?

  When people who do not understand the capabilities, implications, or limitations of a technology attempt to base products on it.
- Can this be solved by "programming"?

  The problems are architectural, because the product (as specified) cannot work. These problems call for architectural solutions.

Deeper

- What does "technological incompetence" encompass?

  Architecture, Design, Coding, Debugging, Testing, and Support.

  Developer productivity is a function of experience, tools, and the problem.

  Often people claim that their programmers are not productive enough, when the real problems are poor design and planning.

Deeper 

- What is the fallacy in:

  You'll know how long it will be when I finish

  If you don't have any idea how long it will take to finish a job, it is probably because you haven't yet figured out what work you have to do. Doing a complicated job without a plan usually ends badly.

Deeper 

- What is the fallacy in:

  If it's late, we can add more people.

  That trick never works:

  Can you find people with the right skills?

  How long will it take to get them on-board?

  Training them slows down the work?

  9 women cannot have a baby in one month!