Can we engineer software?

Webster’s Definitions

en·gi·neer·ing ( n j -nîr ņg) n.
1a. The application of scientific and mathematical principles to practical ends such as the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems.
   b. The profession of or the work performed by an engineer.
2. Skillful maneuvering or direction: geopolitical engineering; social engineering.

Design Under Constraint

“Engineering is design under constraint... Engineering is synthetic - it strives to create what can be, but it is constrained by nature, by cost, by concerns of safety, reliability, environmental impact, manufacturability, maintainability and many other such 'ilities.' ...”

William Wulf and George Fisher
Moore’s “Law”: computing power roughly doubles every 18 months!

Constraints Software Engineers Face

Not like those for “real” engineers:
- Weight, physics, etc.

**Complexity** of what we can understand
- Most important constraints:
  - Limits of human memory
  - Cost of human effort

This class is about managing complexity to efficiently produce reliable complex software systems.

How is engineering software different from engineering bridges?

**Bridges**
- Physical stuff
- Continuous
  - Calculus
  - Testing/analysis is “easy” if the bridge holds for 1M kg, it also probably holds 0.99M kg

**Software**
- Virtual stuff
- Discrete
  - Logic, Discrete Mathematics
  - Testing/analysis is difficult

Requirements are (usually) obvious and easy to describe
A good design is apparent to everyone immediately

...And that, in simple terms, is what’s wrong with your software design.
Course Overview

Obvious when it fails
Bridge makers get sued
Architects need licenses
Sibley & Walker (~30 years between failures)

Software
Falls down quietly (usually)
Software vendors blame user, charge for upgrades
Anyone can make software, no one gets sued

Software Failures
Ariane 5 (1996)
Spanair flight 5022 (2008)

where cs1120 ends...
I think that it’s extraordinarily important that we in computer science keep fun in computing. When it started out, it was an awful lot of fun. Of course, the paying customer got shafted every now and then, and after a while we began to take their complaints seriously. We began to feel as if we really were responsible for the successful, error-free perfect use of these machines. I don’t think we are. I think we’re responsible for something more than when we use of these machines.

Small, Fun Programs (“cs1120“) vs. Big, Important Programs (simulated in “cs2220“)

Small, Fun Programs
Fast enough to finish
Friendly inputs
Keep in memory

Important Programs
Fast enough to satisfy requirements
Unfriendly inputs
Too big

Introductions Thursday
How Big are Big Programs?

Largest program in cs1120: ~1000 lines of code
F-22 Stealth Fighter Avionics Software: 1.5M loc
Linux: 10M lines of code
Windows (XP): ~50M lines of code
Amazon.com: ~100M lines of code
Modern automobile: ~100M lines of code

Typical estimate: $18 per line of code
Typical estimate: 1 bug per 1000 lines of (production) code

Grading

A+: I would be willing to fly in a plane running software you designed and wrote
A: I would be willing to shop in an ecommerce store you built
B: I would trust you to manage programmers working on important software

(See syllabus for grading details.)
Course Summary

Main ideas:

**Abstraction**
Using and designing data abstractions

**Specification**
Understanding and writing declarative specifications

**Analysis**
Static: reasoning about behavior
Dynamic: developing and executing testing strategies

Learn by doing:
5 smallish software projects (problem sets 1-5)
   individually, in small teams, 1-2.5 weeks each
1 larger team project: (almost) anything you want

Expected Background

Prerequisite: cs1120/cs150
You should be able to:

Write and understand **short programs**
Write and understand **recursive definitions**
Use **procedures** as parameters and results
Analyze the **asymptotic running-time** of a procedure
Understand replacement (BNF) **grammars**

If you don’t have this background, you may still be able to take the class (talk to me).

Course History

2002: First offered (cs201j)
    Developed with support from National Science Foundation
Spring 2006: BACS Degree launched
Fall 2006: cs205
Fall 2007, 2008, 2009: cs205 (taught by Paul Reynolds)
Fall 2010: cs2220

Course Pledge

Not the classroom pledge!
The whole point of being at a University is so you can:
   — Learn from your classmates
   — Learn better by teaching your classmates
**READ, sign and return the cs2220 Pledge next class** (Thursday)
If you disagree with anything, this is your chance to object
There may be questions about the pledge on a quiz!

Help Available

**Me:** David Evans
Office hours: Mondays, 1:15-2:30pm
    Thursdays, 11am-noon
**Blog comments:** http://www.cs.virginia.edu/cs2220
    Please use this for things that would be useful for everyone
**Email:** evans@cs.virginia.edu (anytime)

Don’t be afraid to ask for help!

**Assistant Teacher:**
**Web site:** http://www.cs.virginia.edu/cs2220

Almost Everything goes on the web

Charge

**This class is about:**
**Managing complexity:** modularity, abstraction, specification
**Engineering dependability:** analysis, redundancy, design

By 5pm Tomorrow: submit registration survey
**Thursday:** Print, read, and return cs2220 pledge
Beginning of class Tuesday: **Problem Set 1 Due**

If you do not satisfy the prereq for this course but want to stay in it, please talk to me now (or Thursday 11-noon, or arrange another time.)