CS 126: Software Design Studio

Prof. Craig Zilles
TA: Shilpa Subrahmanyam
What is this class about?

- My goals for this class:
  1. Improve your programming productivity by $\geq 3x$
  2. Build your self-sufficiency as a programmer
  3. Introduce you to modern computing environments
  4. Provide you skills getting internships / doing hack-a-thons
  5. Have you build a large project relating to your interests
What is this class NOT about?

- This is NOT a ‘Computer Science’ class
  - This is a programming class
  - (i.e., don’t hate CS even if you hate this class)
- But, this class will help you in your ‘Computer Science’ classes
  - Alleviate the low-level programming struggles
  - You can focus your attention on the big ideas!
What is Programming? (two views)

“The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures....

Yet the program construct, unlike the poet's words, is real in the sense that it moves and works, producing visible outputs separate from the construct itself. [...] The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard, and a display screen comes to life, showing things that never were nor could be.”

— Fred Brooks

Pragmatically, programming is the tool that computer scientists use to collect, analyze, and visualize data, automate tasks, make products, mechanically prove theorems, and build tools. As lawyers write prose and architects build models, programming is the underlying tool of the computer scientist.
Programming is unique

“The gap between the best software engineering practice and the average practice is very wide—perhaps wider than in any other engineering discipline”

— Fred Brooks

“The original study that found huge variations in individual programming productivity ... studied professional programmers with an average of 7 years' experience and found that the ratio of initial coding time between the best and worst programmers was about 20 to 1; the ratio of debugging times over 25 to 1; of program size 5 to 1; and of program execution speed about 10 to 1.”

— Steve McConnell
How do you get better at something?

1. Get lessons from experts

2. Practice

3. Get feedback

4. Metacognition
Course Textbook (part 1)

The definitive book on software construction.

~$30 on Amazon (or $37 for Amazon prime).

Check out reviews on Amazon about how good this book is.
Code Reviews

- Groups of <=6 students + 1 moderator
- Meets 2 hours/week (arranged time)
- Present code that you’ve written in the past week
  - Get feedback on your style & design
  - See other possible designs (pick up ideas)
  - Practice presentation & verbal communication skills
Meta-cognition (or Metacognitive Regulation)

“Regulation of cognition" contains three essential skills:

- **Planning:** appropriate selection of strategies and the correct allocation of resources that affect task performance.
- **Monitoring:** refers to one's awareness of comprehension and task performance
- **Evaluating:** refers to appraising the final product of a task and the efficiency at which the task was performed. This can include re-evaluating strategies that were used.
Course Infrastructure

- Java is a relatively verbose language
- Having a good tool accelerates routine drudgery.
- IntelliJ IDEA is a really good tool

*Actually, much more than just Java*
Learning to help yourself

- Very few programs are written completely from scratch.
  - Most rely heavily on libraries, APIs, and frameworks

- Existing code is person-made and arbitrary
  - No one inherently knows how to interface to it
  - Need to be able to read documentation
    - Google and StackOverflow are your friend

- In this class, we’ll encourage you to help yourself
  - Teach a person to fish, and you feed them for a lifetime.
What are we going to do this semester?

- style, refactoring, code reviews
  - layout, commenting, variable usage and naming, control structures
- test-driven development, testing frameworks, coverage
  - defensive programming, assertions, exception handling
- design, design of routines, object-oriented frameworks
  - design patterns, event-driven programming, MVC
- tools: IDEs, source control, debugging, logging, Unix
- user interface design, prototyping, user testing
- client-server network programming, JSON, SQL

Much of this to be done in the context of mobile apps on Android
To do for Thursday

- Read Ch. 31 of Code Complete (v2)
- Install IntelliJ IDEA on your personal machine
  - Also available on Windows machines in Siebel 0403
- Code up your Assessment problem using Ch. 31 ideas
  - Always use your best programming practices
  - Submit through CS 126 Compass2g
- Get/order the course textbook (Code Complete)
- Get an iClicker and bring it to class on Thursday
- Join the CS 126 piazza