CS 126: Software Design Studio

Prof. G Carl Evans

What is this class about?

- **My goals for this class:**
  1. **Improve your programming productivity by >= 3x**
  2. Build your self-sufficiency as a programmer
  3. Introduce you to modern computing environments
  4. Provide skills for 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 much programming experience do you have?

A. Six months or less

B. Six to 12 months

C. One to two years

D. Two to six years

E. More than six years

How do you get better at something?

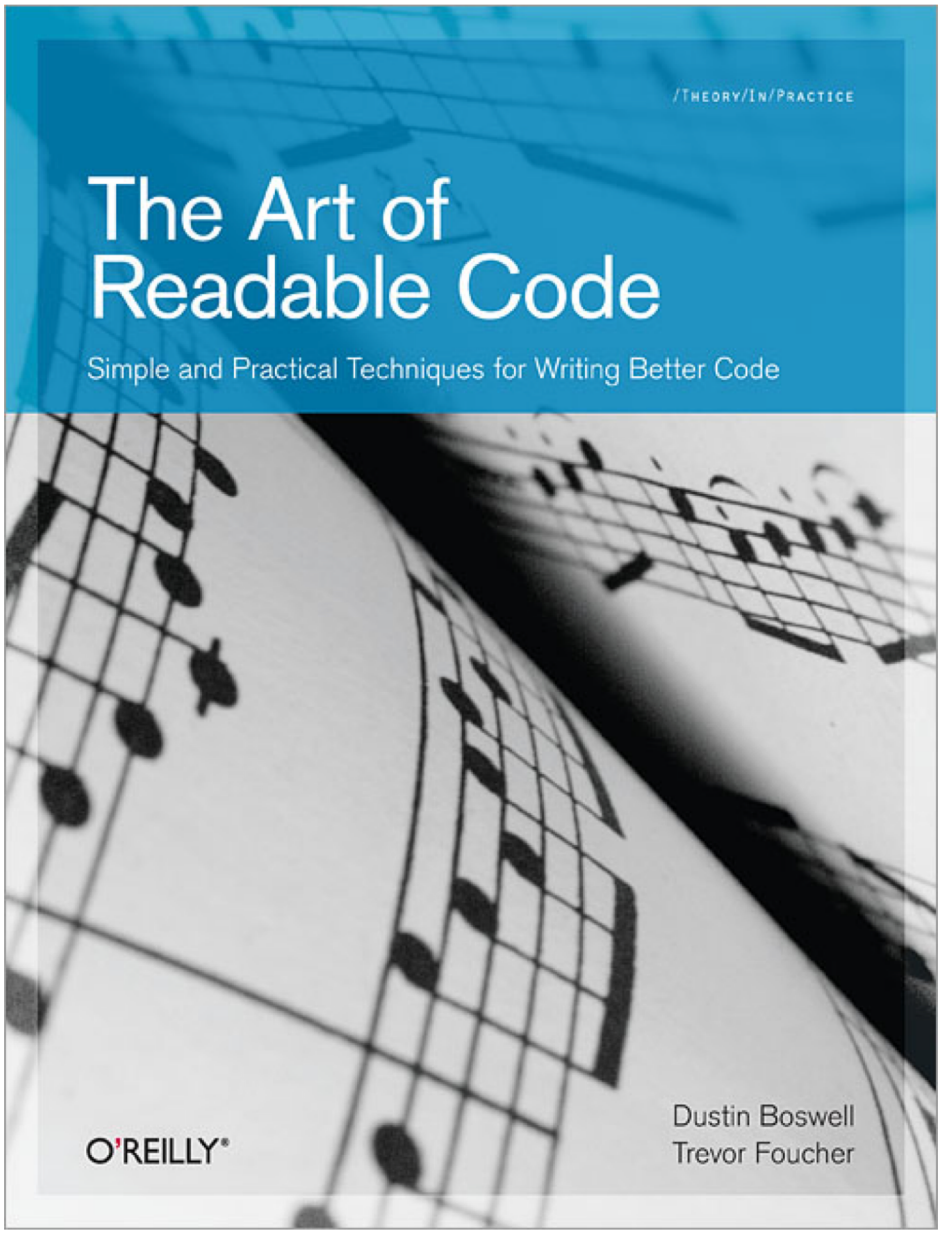
1. Get lessons from experts

2. Practice

3. Get feedback

4. Metacognition

Course Textbook (part 1)



A solid, concise book on software construction.

Less than $30 on Amazon prime (or $16 for a Kindle version).

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.

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**

Course Infrastructure (1)

- Java is a relatively verbose language
- Having a good tool accelerates routine drudgery.
- IntelliJ IDEA is a really good tool (basis for Android Studio)

Course Infrastructure (2)

- **Version control systems (VCS):**
  - A practice that tracks and provides control over changes to a collection of documents/files.
  - Allows access to any prior version.
  - Facilitates collaboration between multiple developers.
- **Git: an industry-standard distributed VCS**
  - You'll use this for developing/submitting your code.
  - Very sophisticated tool; we'll use a subset of features

VCS concepts

- **Repository: A collection of files under version control, along with all of their previous (committed) versions.**
- **Checkout (verb): To make a working copy on your local machine for editing/testing.**
- **Commit (verb): To take a set of file modifications and add them to the repository, usually with a descriptive message.**
- **Commit (noun): The set of changes (a "diff") along with its descriptive message resulting from a commit (verb).**

Git concepts

- **Local repository vs. remote repository:**
  - Git lets you have as multiple related repositories on different machines.
- **Clone:**
  - Make a local repository from a remote repository.
- **Staged:**
  - Files whose changes are to be committed.
- **Push:**
  - Copying local commit to remote repository.
- **Pull:**
  - Bringing changes from remote to local repository. Implemented by a “fetch” then a “merge”.

Version Control, why do we care?

Single Developer:

- **Most things worth doing are too big to do all at once.**
- **Break large projects into small steps:**
  - Design, implement, test, debug, **commit** each step.
  - Have access to every working version through VCS
  - If things stop working:
    - Can inspect the differences between current and last working versions.
    - Can always revert back to last working version (e.g., throw away changes)

Multiple developers:

- **Coordinate edits to a shared set of files**

Java