CSE 331
Software Design & Implementation

Hal Perkins

Winter 2018

Lecture 1 – Introduction & Overview

(Based on slides by Mike Ernst, Dan Grossman, David Notkin, Hal Perkins, Zach Tatlock)

Welcome!

We have 10 weeks to move well beyond novice *programmer*:

- Larger programs
  - Small programs are easy: "code it up"
  - Complexity changes everything: "design an artifact"
  - Analogy: using hammers and saws vs. making cabinets (but not yet building houses)
- Principled, systematic software: What does "it's right" mean? How do we know "it's right"? What are best practices for "getting it right"?
- Effective use of languages and tools: Java, IDEs, debuggers, JUnit, JavaDoc, git, …
  - Principles are ultimately more important than details
    - You will forever learn details of new tools/versions

Concise to-do list

Before next class:

1. Familiarize yourself with website, do readings
2. Read syllabus and academic-integrity policy
   (know this stuff – do we need a quiz?)
3. Go to any section tomorrow if not registered yet
   - If still trying to add, watch for magic URL with form to fill out at end of class
4. Do Homework 0 (see web calendar), due by 10 am Friday! & no late days this time
   - (send mail to cse331-staff if not registered so we can add you to the gradescope course roster to turn in the assignment)

Who: Course staff

- Lecturer:
  - Hal Perkins: Faculty since last millennium, CSE331 veteran
- TAs:
  - Alexey Beall
  - Natalie Fetsch
  - Weifan Jiang
  - Martin Kellogg
  - Jason Qiu
  - Laura Vonessen
  - Yifan Xu
  - Yiyang Xu
  - Nate Yazdani
  - Sarah Yu
- Office hours: will schedule lots. Will try to get started shortly.

Get to know us!

- Make sure this *feels like* a 30-person class with 150 students
- We're here to help you succeed

What's new/different

- First time CSE 331 has had two lectures in the same quarter
  - It's one big class – everyone does the same work & we will try to keep both lectures and all sections in synch
  - OK to attend different lecture occasionally
  - Please try to attend your registered section

- Midterm and final exams will be held outside normal times so everyone can take the exams together
  - Will try to pin this down soon, but don't have it yet.

Acknowledgments

- Course designed/created/evolved/edited by others
  - Michael D. Ernst
  - Dan Grossman
  - David Notkin
  - (me)
  - Zach Tatlock
  - Kevin Zatloukal
  - Several dozen amazing TAs
- Hoping my own perspective offers benefits
- [Because you are unlikely to care, I won't carefully attribute authorship of course materials]

Staying in touch

- Message Board
  - Using Google groups this quarter (link on course home page by later today)
  - Join in! Use for most discussions; staff will read/contribute
  - Help each other out and stay in touch outside of class

- Course staff: **`cse331-staff@cs.washington.edu`**
  - For things that don't make sense to post on message board
  - Please do *not* send messages to individual TAs – we need to get the traffic centrally so we can route it appropriately

- Course email list: **`cse331a_wi18@u.washington.edu`**
  - Students and staff already subscribed (your UW email address)
  - You must get announcements sent there
  - Fairly low traffic

Lecture and section

- Both required

- All materials posted, but they are visual aids
  - Arrive punctually and pay attention (& take notes!)
  - If doing so doesn't save you time, one of us is messing up (!)

- Section will often be more tools and homework-details focused
  - Especially next week: preparing for projects
  - Watch for room changes – might be different by tomorrow

- Other posted handouts related to class material

Laptops in class?

- Just say no!!
- No open laptops during class (no kidding!)
  - (unless we're doing something where we need them)
- Why? You will learn better if you are mentally present during the class (not just physically)
- Got the urge to search? Ask a question!
- Exception (maybe): if you *actually*, *really*, ***really***, use a tablet to take notes, ok, but sit in the back row so you don't distract
  - But you'll learn more if you use paper instead! (really!!)
- And no phone texting, web surfing, etc., either…
- You may close your electronic devices now
  - And take out a piece of paper to write notes ☺

Homeworks

- Biggest misconception about CSE331 (?)

  "Homework was programming projects that seemed disconnected from lecture"

  - If you think so, you are making them harder!
    - Reconsider
    - Seek out the connections by thinking-before-typing
    - Approaching them as CSE143 homework won't work well
    - Don't keep cutting with a dull blade

- First couple assignments are "more on paper", followed by software development that is increasingly substantial

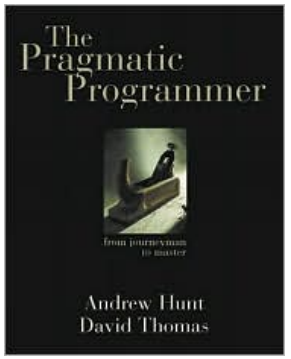
- Four (**4**) late days for the quarter: save for emergencies
  - **Max 2** per homework, save them for later
  - No partial credit / extended deadlines / etc. after that

Resources – Books

Required:

- *Pragmatic Programmer*, Hunt & Thomas
- *Effective Java* 2nd ed, Bloch
  - Or 3rd ed when out this Saturday!

Serious programmers
should study these



Decent "Java book" is a wise thing to have

- *Core Java* Vol I, Horstmann, or
- *Java 9 for the Impatient*, Horstmann

And must learn to use the Java API Docs

Readings (and quizzes)

- These are “real” books about software, approachable in 331
  - Occasionally slight reach: accept the challenge
- Overlap only partial with lectures
- Want to make sure you “do it”
  - Reading and thinking about software design is essential
    - Books may seem expensive given your budget, but very cheap as a time-constrained professional
  - Will have some simple online reading quizzes
    - Might be in a few batches; no late days
  - Material is fair-game for exams

Books? In the 21st century?

- Why not just use Google, Stack Overflow, Reddit, Quora, …?
- Web-search good for:
  - Quick reference (What is the name of the function that does … in Java? What are its parameters?)
  - Links to a good reference
- (can be) Bad for
  - Why does it work this way?
  - What is the intended use?
  - How does my issue fit into the bigger picture?
- Beware:
  - Random code blobs cut-and-paste into your code (why does it work? what does it do?)
  - "This inscrutable incantation solved my problem on an unknown version for no known reason"

Exams

- Midterm: middle of the quarter – probably late afternoon; will try to announce as quickly as possible
- Final: Official times are Monday March 12 at 8:30 and 2:30; but planning to reschedule to get a single time, maybe on a different day (hopefully *not* 8:30)
- All the concepts, different format than homework
  - Will post old exams from various quarters later

Academic Integrity

- Read the course policy carefully
  - Clearly explains how you can and cannot get/provide help on homework and projects
- Always explain any unconventional action
- I have promoted and enforced academic integrity since I first started teaching (many years ago)
  - Great trust with little sympathy for violations
  - Honest work is the most important feature of a university (or engineering or business or life). Anything less disrespects your colleagues (including course staff) and yourself.

Questions?

Anything I forgot about course mechanics before we discuss, you know, software?

Goals

- CSE 331 will teach you to how to write correct programs
- What does it mean for a program to be correct?
  - Specifications
- What are ways to achieve correctness?
  - Principled design and development
  - Abstraction and modularity
  - Documentation
- What are ways to verify correctness?
  - Testing
  - Reasoning and verification

Main topic: Managing complexity

- Abstraction and specification
  - Procedural, data, and control flow abstractions
  - Why they are useful and how to use them
- Writing, understanding, and reasoning about code
  - Will use Java, but the issues apply in all languages
  - Some focus on object-oriented programming
- Program design and documentation
  - What makes a design good or bad (example: modularity)
  - Design processes and tools
- Pragmatic considerations
  - Testing
  - Debugging and defensive programming
  - [more in CSE403: Managing software projects]

The goal of system building

- To create a correctly functioning artifact
- All other matters are secondary
  - Many of them are ***essential*** to producing a correct system
- We insist that you learn to create correct systems
  - This is hard (but fun and rewarding!)

Related skill: *communication*

- Can you convince yourself and others something is correct via precise, coherent explanations?

Why is building good software hard?

- Large software systems are enormously complex
  - Millions of "moving parts"
- People expect software to be malleable
  - After all, it's "only software"
- We are always trying to do new things with software
  - Relevant experience often missing

- Software engineering is about:
  - Managing complexity
  - Managing change
  - Coping with potential defects
    - Customers, developers, environment, software

Orders of magnitude

"We understand walls in terms of bricks, bricks in terms of crystals, crystals in terms of molecules etc. As a result the number of levels that can be distinguished meaningfully in a hierarchical system is kind of proportional to the logarithm of the ratio between the largest and the smallest grain, and therefore, unless this ratio is very large, we cannot expect many levels.

"In computer programming our basic building block has an associated time grain of less than a microsecond, but our program may take hours of computation time. I do not know of any other technology covering a ratio of 10^{10} or more: the computer, by virtue of its fantastic speed, seems to be the first to provide us with an environment where highly hierarchical artifacts are both possible and necessary."

-- Dijkstra

Programming is hard

- It is surprisingly difficult to specify, design, implement, test, debug, and maintain even a simple program

- CSE331 will challenge you

- If you are having trouble, *think* before you act
  - Then, look for help

- We strive to create assignments that are reasonable if you apply the techniques taught in class…
  … but likely hard to do in a brute-force manner
    … and almost certainly impossible to finish if you
      put them off until a few days before they're due

Prerequisites

- Knowing Java is a prerequisite
  - We assume you have mastered CSE142 and CSE143

Examples:

- Sharing:
  - Distinction between `==` and `equals()`
  - Aliasing: multiple references to the same object
- Object-oriented dispatch:
  - Inheritance and overriding
  - Objects/values have a run-time type
- Subtyping
  - Expressions have a compile-time type
  - Subtyping via `extends` (classes) and `implements` (interfaces)

You have homework!

- Homework 0, due online by 10 am Friday (no late days)
  - Write (don't run!) an algorithm to rearrange (swap) the elements in an array
  - in O(*n*) time (and preferably in a single pass)
  - And argue (prove) in concise, convincing English that your solution is correct!

- Purpose:
  - Great practice
  - Surprisingly difficult
  - So we can build up to reasoning about large designs, not just 5-10 line programs

Written homework submission

- Trying something new this quarter: gradescope
- Sometime late today or tomorrow you'll get mail from gradescope.com with login details (id = UW email address)
  - (If not registered, send mail to cse331-staff with your name, UW email, and UW student # so we can create an account)
- Then click on the link or follow the one on the course home page, upload your file, and identify which pages have answers to which questions
- You'll get email when grades are available

CSE331 is hard! (but very rewarding)

- You will learn a lot!
- Be prepared to work and to think
- The staff will help you learn
  - And will be working hard, too

- So let's get going…
  - Before we create masterpieces we need to hone our ability to reason very precisely about code…

A Problem

"Complete this method such that it returns the largest value in the first **`n`** elements of the array **`arr`**."

```
int max(int[] arr, int n) {
    ...
}
```

A Problem

"Complete this method such that it returns the largest value in the first **`n`** elements of the array **`arr`**."

```
int max(int[] arr, int n) {
    ...
}
```

What questions do you have about the *specification*?

Given a (better) specification, is there exactly 1 *implementation*?

Moral

- You can all write the code

- More interesting in CSE331:
  - What if n is 0?
  - What if n is less than 0?
  - What if n is greater than array length
  - What if there are “ties”?
  - Ways to indicate errors: exceptions, return value, …
  - Weaker versus stronger specifications?
  - Hard to write English specifications (n vs. n-1)

Concise to-do list

Before next class:

1. Familiarize yourself with website, do readings
2. Read syllabus and academic-integrity policy
3. Go to any section tomorrow if not registered yet
4. Do Homework 0 (see web calendar), due before 10 am Friday! (no late days this time)
   - (send mail to cse331-staff if not registered so we can add you to the gradescope course roster to turn in the assignment)

If still trying to register, copy down the URL that's coming up next…