CSE 331
Software Design and Implementation

Lecture 1
Introduction

Leah Perlmutter / Summer 2018

Overview

- Motivation
- Introductions
- Course Philosophy
- Administrivia
- 331
- Closing Announcements

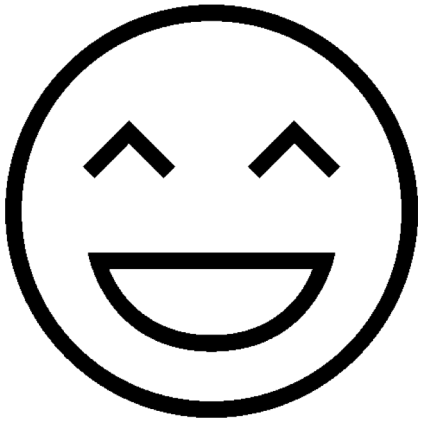
Motivation

Writing *Good* Code

- What are some properties of “good code”?
  - Easy to reason about `understandable`
  - Easy for your collaborators to reason about
  - Easy to debug `debuggable`
  - Easy to extend `extensible`
  - **Elegant**
  - Many more...

Elegance

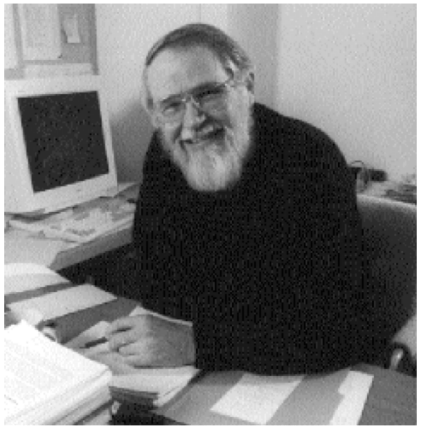
- Elegant code might give you **Shivers Of Joy**



- Inelegant code might give you **Shudders of Revulsion**

Controlling Complexity

"Controlling complexity is the essence of computer programming."



-- Brian Kernighan

(UNIX, AWK, C, …)

Controlling Complexity

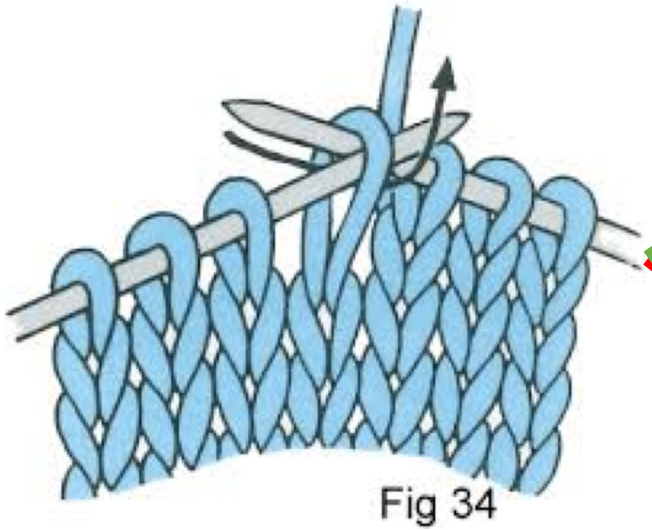
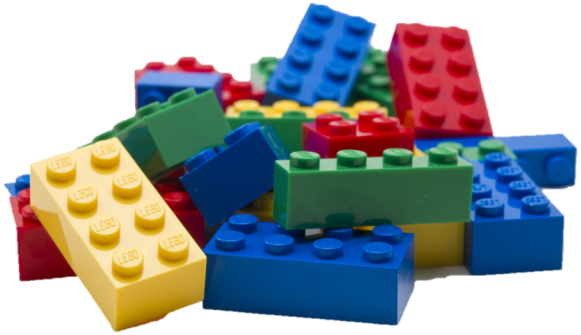
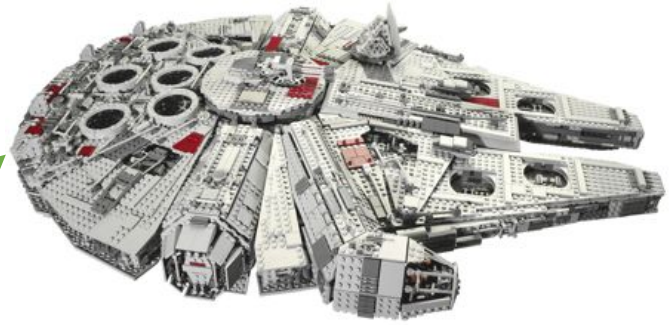


Fig 34

Controlling Complexity





Overview

- ☑ Motivation
- ❑ Introductions
- ❑ Course Philosophy
- ❑ Administrivia
- ❑ 331
- ❑ Closing Announcements

Welcome!

Instructor

- Leah Perlmutter
- Pronouns: she, her
- 3rd year PhD student
- Worked in software industry before PhD
- TA’d 331 last quarter
- This is my first time teaching!

TAs

Haiqiao Chen

JongHo Lee

Wei Liao

Frank Qin

Matt Xu

Joyce Zhou

Students

- Welcome and thanks for being here!!!!!
- 9 different majors represented
  - 27 Computer Science majors
  - 8 Pre-science, Pre-engineering, or Pre-major
  - 7 In other majors or non-matriculated
- Learning names

Students

1. Legal Name
2. Your pronouns
3. Preferred name
4. Pronunciation of your preferred name (optional)
5. One thing you're looking forward to in 331
6. A fun fact about yourself!

1. Leah Perlmutter
2. she, her
3. Leah
4. LAY-uh
5. Students' "aha" moments.
6. I love mountain biking and climbing!

Overview

- ☑ Motivation
- ☑ Introductions
- ❑ Course Philosophy
- ❑ Administrivia
- ❑ 331
- ❑ Closing Announcements

Course Philosophy

Active Learning

- Learning is an active process, not only for the teacher, but also for the student!
- Ways to put the ideas through your brain
  - Note taking
  - Asking questions
  - Example problems and worksheets in class
  - Discussing or re-explaining ideas to a peer
  - What are your strategies?
    - Can be existing strategies or something you'd like to try for the first time in this class!

Collaboration

- You are encouraged to discuss the material with peers!
- You are required to report the names of your collaborators (or “none”) on every homework assignment
- See Collaboration Policy on the course website
- The spirit of this policy is to help you learn

Struggling

- It's normal for good students to struggle on CSE 331 homeworks!
- Don't give up
- Do discuss with peers
  - Other students likely share your struggle
- Do persist and keep thinking about the problem
- Do seek help if you are struggling *unproductively*

Overview

- ☑ Motivation
- ☑ Introductions
- ☑ Course Philosophy
- ❑ Administrivia
- ❑ 331
- ❑ Closing Announcements

Administrivia

Communication

- Course website
- Message board (Google Groups)
  - Venue for nearly all questions and answers
- Staff email list: `cse331-staff [at] cs.washington.edu`
  - For special circumstances
  - Remember to always reply-all!
- Student email list: `cse331a_su18 [at] uw.edu`
  - You will receive occasional important announcements on this list

Grade Breakdown

- 60% Homework (10 assignments)
  - First three are written assignments
  - Last seven are coding
- 15% Midterm
- 15% Final (not cumulative)
- 5% Reading quizzes
- 5% Participation

Homework (60%)

- Written assignments (HW0, 1, 2) submitted electronically via Gradescope
  - Use your @uw.edu email address
  - More instructions on course website
- Coding assignments (HW3, 4, 5, 6, 7, 8, 9) submitted electronically via Gitlab
  - More info on this later
- Homework is weighted according to the number of points in the assignment

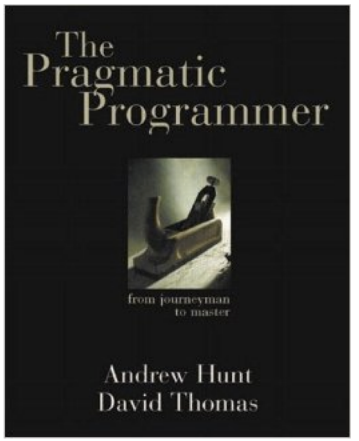
Exams (15% + 15%)

- Midterm and final equally weighted
- Will be held during lecture time
- Final will test the second half of the material
  - (No finals week in summer quarter)

Readings and Quizzes (5%)

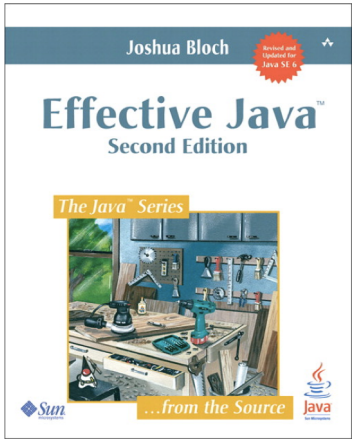
The Pragmatic Programmer

- Hunt and Thomas (1999)
- Collection of best practices



Effective Java

- Bloch, 2nd edition (2000)
- Bloch, 3rd edition (2017)
- OOP design, expert tips



Readings and Quizzes (5%)

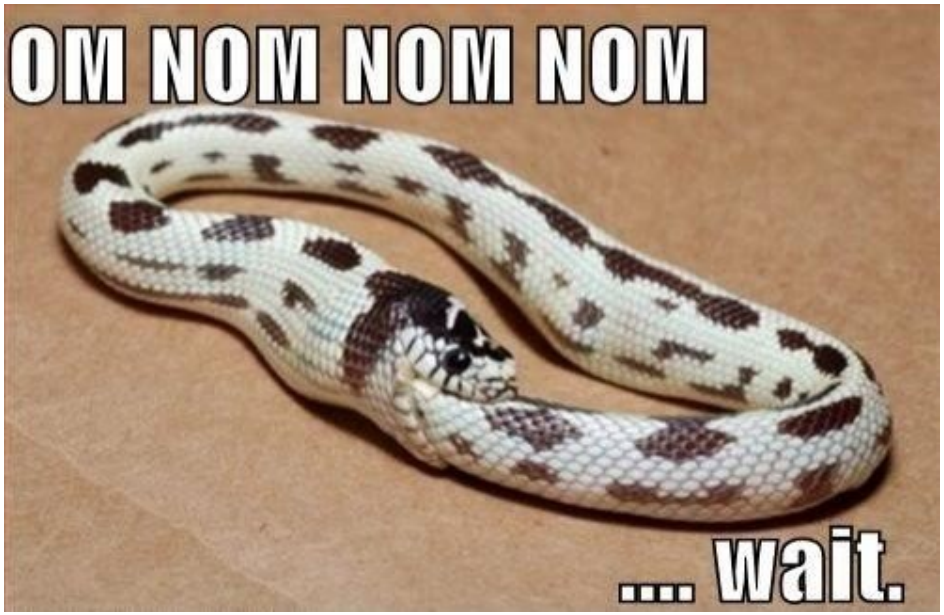
- Readings are related to lecture and homework!
- Try to make connections as you read and as you attend lecture
- There may be a 1-week offset between reading material and lecture material
- Reading quizzes will be via Google Forms, and the quiz links will be posted on the course website

Participation (5%)

- The following will positively affect your participation grade
  - Asking questions in class
  - Coming to instructor office hours
  - Reporting collaborators on homework
    - See Collaboration Policy on course website

Attendance

- Make the most of your tuition dollars
- Do not self-cannibalize by skipping lecture or section to do this class's homework



Academic Integrity

- Carefully read the policy on the course website
- Do not seek help in a way that destroys the intellectual challenge of the course!
- Honest work is the foundation of UW / academia
  - Your fellow students and I trust you deeply
  - Zero tolerance for violations, can end career

Organization

331 is a big, complex machine.

As a first-time instructor, I'm a bit scared of 331 too, so I'll need your help figuring out what works for all of us.

Patience and good faith much appreciated!

Feedback

- I really value your feedback about this course!
- You have the ability to shape me as an instructor!
- Feedback mechanisms
  - Instructor office hours
  - Staff email list
  - Mid-course survey
  - Mid-course external evaluator session
  - End of quarter course evaluations

Overview

- ☑ Motivation
- ☑ Introductions
- ☑ Course Philosophy
- ☑ Administrivia
- ❑ 331
- ❑ Closing Announcements

331

Goals

One focus will be writing *correct* programs

What does it mean for a program to be correct?

- It must match its *specification*

How can we determine if a program is correct?

- Testing, Model Checking, Verification (proof)

What are ways to build correct programs?

- Principled design and development
- Abstraction, modularity, documentation

Controlling 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

- Use Java, but the principles apply broadly
- 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 construct a correctly functioning artifact

All other considerations are secondary

- Though many required to produce a correct system

Learning how to build correct systems is *essential* and very difficult, but also fun and rewarding.

Why is Good Software Hard?

Software is different from other artifacts

- We build general, reusable mechanisms
- Not much repetition, symmetry, or redundancy
- Large systems have millions of complex parts

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 artefacts are both possible and necessary.

-- Dijkstra

Why is Good Software Hard?

Software is expected to be malleable

- You can't download a new chip into your phone
- But you can update web pages, apps, and the OS
- Aggressive competition for more features, platforms
- Requirements, laws, and companies change

We are pioneers and explorers!

- Often writing a new kind of system
- Little relevant experience or specific theory

Software engineering is about:

- Managing complexity, managing change
- Coping with potential defects: users, devs, environment

Programming is Hard

Despite decades of research, still surprisingly difficult to specify, design, implement, test, and maintain even small, simple programs.

Our assignments will be reasonable if you apply the techniques taught in class…

... but likely very difficult to do brute-force

... and almost certainly impossible unless you start very early.

If you're frustrated, *think* before you type!

Prerequisites

Knowing Java is essential

- We assume you've mastered 142, 143

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 PM Wednesday

- Rearrange array elements by sign
- O(n) time, preferably in a single pass
- Only write (don’t run!) your algorithm
- Clearly and concisely prove your solution correct!
- Full assignment posted on course website...

Purpose:

- Great practice and warm-up
- Surprisingly difficult
- Working up to reasoning about large designs

CSE 331 is a Challenge

We are going to learn a lot and have a good time

Be prepared to work hard and think hard

- It's normal to struggle on the homework!

The staff is here to help you learn

- We will be working hard too!

So, let's get to it!

- Before we create masterpieces, we first need to hone our ability to reason about code…

A Problem

"Complete this method so that it returns the index of the max of the first **`n`** elements of the array **`arr`**."

```
int index_of_max(int[] arr, int n) {
   …
}
```

A Problem

"Complete this method so that it returns the index of the max of the first **`n`** elements of the array **`arr`**."

```
int index_of_max(int[] arr, int n) {
   …
}
```

What should we ask about the *specification?*

Given (better) specification, how many possible implementations are there?

Moral

You can all write this code

More interesting for us in 331:

- What if n is 0?
- What if n is less than 0?
- What if n is greater than the array length?
- What if there are “ties”?
- How should we indicate error:
  - exception, return value, fail-stop, …
- Weaker vs. stronger specs?
- Challenge writing English specs (n vs. n-1)

Something to Chew On

What is the relationship of "goodness" to "correctness" for programs?

Announcements

To Do

- Check out the course website
  - Read syllabus, academic integrity policy, collaboration policy, and gradescope instructions
- Log into the course message board
- HW0 is due Wednesday
  - Posted on course website
  - Log into Gradescope today to make sure it works
- Reading quiz 1 (37 pages) is due Thursday
  - Get your textbooks now!