CSE 331
Software Design and Implementation

Lecture 1
Introduction

Zach Tatlock / Spring 2018

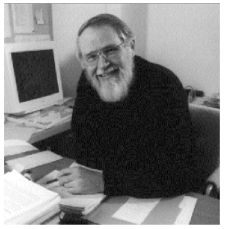
The Big Picture

Welcome!

10 week study of the craft of programming

How do we build good programs?

"Controlling complexity is the essence of computer programming."



-- Brian Kernighan
(UNIX, AWK, C, …)

Controlling Complexity

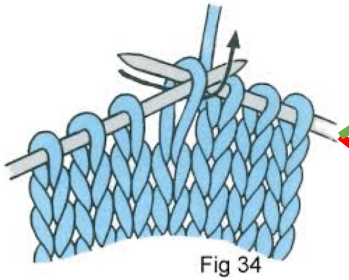
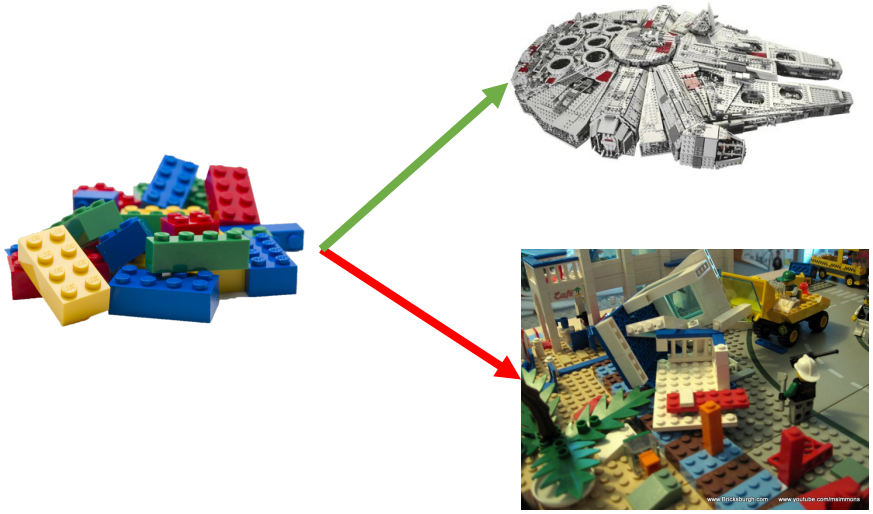


Fig 34

Controlling Complexity



Learning to Control Complexity

First, we need to refine our goals:

- What quality makes a program good?
- How can we tell if a program is good?
- How do we build good programs?

To answer, we'll learn *principles* and use tools:

- Modularity, documentation, testing, verification
- Tools: *Java, IDEs, debuggers, JUnit, JavaDoc, git*

Tools change, principles are forever.

Administrivia

Course Staff

Instructor:

Zach Tatlock
ztatlock@cs
CSE 546

Researcher in verification, compilers, systems.

Course Staff

TAs:

Alexey Beall	Leah R Perlmutter
Tim Chirananthavat	Kevin Pusich
Hongtao Huang	Chen (Jason) Qiu
Weifan Jiang	Yifan (Vanadis) Xu
Cody Kesting	

Office hours settling in over next couple days.

Several 331 veterans = expert guides!

Course Staff

Credits

Great course material based on work by:

- Michael Ernst
- Hal Perkins
- Dan Grossman
- David Notkin
- Dozens of amazing TAs
- Hundreds of incredible students (you!)

Staying In Touch

We'll use Google Groups:

https://goo.gl/2HVrrQ

- Discuss HW, lecture, readings
- Be nice, be **"PROFESSIONAL"** (staff will monitor)
- Ask *good questions*, give helpful answers
  http://www.catb.org/esr/faqs/smart-questions.html

Critical / emergency updates to list:

cse331a_sp18 [at] uw.edu

- Mandatory, but expect very low traffic

Lecture and Section

Both required

- Arrive punctually, ask questions, take notes
- Your participation is crucial for everyone's success

Materials will be posted, but just visual aids

Section often more focused on HW and tools

- This week: more detail on Lecture 2 concepts
- Next week: preparing for projects

Homework

Common 331 misconception:

"Homework seemed disconnected from lecture."

If it feels that way, you're making them harder!

- Reconsider and seek out connections to lecture
- Do think carefully before typing
- Do not keep cutting with a dull blade

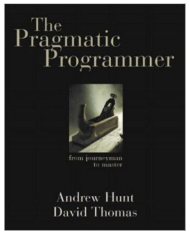
Early assignments are "on paper", followed by increasingly substantial software development.

4 late days, max 2 per assignment, use wisely!

Text Resources

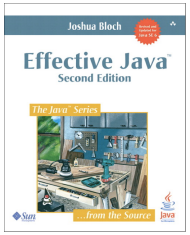
The Pragmatic Programmer

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



Effective Java

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



Java API Docs:

http://docs.oracle.com/javase/8/docs/api/

Readings and Quizzes

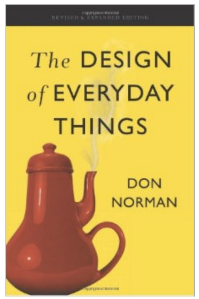
These are real programming books

- Hard-won advice from top-notch hackers
- Stuff all serious programmers should know
- Approachable but sometimes challenging
- Only partial overlap with lecture

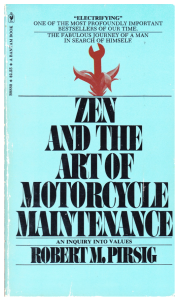
Quizzes to ensure you keep up with reading

- Reading and contemplating design is essential
- Time investment that pays dividends in the long run
- Material may be on exams

Other Reading



The Design of Everyday Things



Zen and the Art of Motorcycle Maintenance

Not directly used in 331, but worthwhile reads.

Reading Books in 21st Century

Google, Stack Overflow, Reddit, etc. good for:

- Quick reference and debugging
- Links to more in-depth treatment of a topic

Search often less good for:

- Why did this bug arise? How could it be avoided?
- Why is the system designed this way? Alternatives?

Beware copy-paste coding

- Security vulns have propagated through forums
- See The Full StackOverflow Developer

Exams

Midterm: TBD, roughly week 5
in class

Final: Wednesday, June 6
8:30 – 10:20 (sorry!)

Can cover any concepts from the course

- Different format than homework
- Will post past exams from various instructors

Academic Integrity

Carefully read course policy

- Clearly explains how you can / cannot get help on homework and projects

Always explain any unconventional action

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.

Even after a couple times through the gauntlet, I'm still 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!

TODO

1. Log into the 331 Google Group
2. Check out the course website

   http://cs.washington.edu/331
3. Read syllabus and academic integrity policy
4. Do Homework 0 by Friday 5PM!
   - Submit to Gradescope

Questions?

Anything I forgot before we discuss, you know, software?

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.

Related skill: *communication*

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

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 5 PM Friday

- 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!

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

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?

TODO

1. Log into the 331 Message Board
2. Check out the course website

   http://cs.washington.edu/331
3. Read syllabus and academic integrity policy
4. Eat your vegetables
5. Go to Section on Thursday
6. Do Homework 0 by Friday 5 PM!
   - Submit via Gradescope