CSE 331
Software Design & Implementation

Kevin Zatloukal
Spring 2020
Lecture 1 – Introduction & Administrivia
(Based on slides by Mike Ernst, Dan Grossman, and many others)
Motivation
What is the goal of CSE 331?

How to build harder-to-build software

• Move from CSE 143 problems toward what you’ll see in upper-level courses and in industry

Specifically, how to write code of

• Higher quality
• Increased complexity

We will discuss tools and techniques to help with this and the concepts and ideas behind them

– There are timeless principles to both
– Widely used across the industry
What is high quality?

Code is high quality when it is

1. Correct
   - Everything else is of secondary importance

2. Easy to change
   - Most work is making changes to existing systems

3. Easy to understand
   - Needed for 1 & 2 above
How do we ensure correctness...

... when **people** are involved?

People have been known to

- walk into windows
- drive away with a coffee cup on the roof
- drive away still tied to gas pump
- lecture wearing one brown shoe and one black shoe
What is increased complexity?

Analogy to building physical objects:
• 100 well-tested LOC = a nice cabinet
• 2,500 LOC = a room with furniture
• 2,500,000 LOC = 1000 rooms ≈

North Carolina class WW2 battleship
≈

the entire British Naval fleet in WW2
Actually, software is more complex…

• Every bit of code is unique, individually designed
  – US built 10 identical Essex carriers

  – Software equivalent would be one carrier 10 times as large:

• Defects can be even more destructive
  – A defect in one room can sink the ship
  – But a defective OS could sink the *whole fleet*
Scale makes everything harder

Modularity makes scale **possible** but it’s still **hard**…

- Time to write N-line program grows faster than linear
  - Good estimate is $O(N^{1.05})$ [Boehm, ‘81]
- Bugs grow like $\Theta(N \log N)$ [Jones, ‘12]
  - 10% of errors are between modules [Seaman, ‘08]
- Communication costs dominate schedules [Brooks, ‘75]
- Small probability cases become high probability cases
  - Corner cases are more important with more users

**Corollary**: quality must be even higher, per line, in order to achieve overall quality in a *large* program
People Do Build Great Software

Full scope of the challenge:

- software is built by people, who make mistakes all the time
- surprisingly difficult to get even a small program to work
- needed to write hundreds of millions of lines of code
- each line gets harder to write as the program scale

Despite those challenges, we have lots of software that works

- hundreds of millions of lines of working programs
- products rarely fail because the software is too buggy

How do we do it?
How do we ensure correctness...

... when **people** are involved?

People have been known to
- walk into windows
- drive away with a coffee cup on the roof
- drive away still tied to gas pump
- lecture wearing one brown shoe and one black shoe

**Key insights:**
- Can’t stop people from making mistakes
- Can stop mistakes from getting to users
How do we ensure correctness?

Best practice: use three techniques (we’ll study each)

1. **Tools**
   - Type checkers, test runners, etc.

2. **Inspection**
   - Think through your code carefully
   - Have another person review your code

3. **Testing**
   - Usually >50% of the work in building software

Each removes ~2/3 of bugs. Together >97%
How do we cope with complexity?

We tackle complexity with **modularity**

- Split code into pieces that can be built independently
- Each must be documented so others can use it
- Also helps understandability and changeability
What is high quality code?

In summary, we want our code to be:

1. Correct
2. Easy to change
3. Easy to understand
4. Easy to scale (modular)

These qualities also allow for increased complexity
What we will cover in CSE 331

- Everything we cover relates to the 4 goals
- We’ll use Java but the principles apply in any setting

**Correctness**

1. Tools
   - Git, IntelliJ, JUnit, Javadoc, …
   - Java libraries: equality & hashing
   - Adv. Java: generics, assertions, …
   - debugging
2. Inspection
   - reasoning about code
   - specifications
3. Testing
   - test design
   - coverage

**Changeability**

- specifications, ADTs
- listeners & callbacks

**Understandability**

- specifications, ADTs
- Adv. Java: exceptions
- subtypes

**Modularity**

- module design & design patterns
- event-driven programming, MVC, GUIs
Administrivia
Who: Course staff

• **Instructor:** Kevin Zatloukal (kevinz at cs)
  – 15 years in industry, 5th year teaching

• ≈15 great TAs
  – mix of veterans and new

• Office hours posted soon
  – (starting later this week)

*Get to know us!*
  – We’re here to help you succeed
Who: Students

- Assuming you have mastered CSE142 and CSE143
- Hoping (but not assuming) have you taken 311
  - will connect to 311 material where it arises
- Assuming you are in your second year of CS courses
  - seniors may be bored
Prerequisites

- Knowing Java is a prerequisite

Examples:
- Difference between `int` and `Integer`
- Distinction between `==` and `equals()`
- Aliasing: multiple references to the same object, what does assignment (`x=y;`) really mean?
- Subtyping via `extends` (classes) and `implements` (interfaces)
- Method calls: inheritance and overriding; dynamic dispatch
- Difference between compile-time and run-time type
Unique Situation (for all of us)

- Much of the rest of this is **subject to change**
  - but we are learning as we go

- Personal issues may arise
  - let me know
  - we will make accommodations as much as possible
Staying in touch

• Ed message board (link on course web page)
  – should have received an invitation already
  – best place to ask questions

• Course staff: cse331-staff@cs.washington.edu
  – For things that don’t make sense to post on message board

• Course email list: cse331a_sp20@u.washington.edu
  – Students already subscribed (your UW email address)
  – You must get announcements sent there
  – Fairly low traffic – one way (from staff to everyone)
Lectures

• Includes a pre-recorded video and a live session

• Each pre-recording posted 2 days before
  – please watch that portion beforehand

• Each live session will each be a little different
  – some lecture, Q&A, problems, work in small groups, etc.
  – link to recording in Canvas
  – slides posted on web site
Section

- Will be focused on **helping with homework**
  - held on day HW is released
  - get you started with the work to be done
  - they should be very useful

- Live via Zoom video
  - links on Canvas (see Zoom app)

- Aiming to have 10 sections with 16 students each
  - will split time schedule sections into two parts
  - details coming soon
Homework Assignments

• Roughly 1 assignment per week

• First 3 are paper assignments
  – submit these in Gradescope
  – should get an invite email before Tuesday
    • let me know if you don’t

• Remaining 7 are coding assignments
  – generally due on Wednesday by 11pm
  – submit and tag your code in Gitlab
    • TAs will grade and get feedback to you
Homework Assignments

• Biggest misconception (?) about CSE331
  “Homework was programming projects that seemed disconnected from lecture”

• If you think so, you are making them harder!
  – approaching them as CSE143 homework won’t work well
  – each HW designed to teach topics from prior lectures
  – seek out the connections by before typing

• (Tip: this is also true of exams / quizzes)
Late Policy: Written Assignments

- Allowed only in special situations
  - let us know 36 hours beforehand
  - will also make exceptions for emergencies
Late Policy: Coding Assignments

• Same special situations as written assignments

• And also:
  – Up to 4 times this quarter you can turn in a homework assignment one day late
  – Not accepted for credit after that.
  – Late days are 24-hour chunks

• Why?
  – keep you on schedule (real world has deadlines)
  – get feedback to you before next deadline
Academic Integrity

• “The code you submit must be your own”
  – no copying from other students, web pages, etc.

• Read the full course policy carefully
  – ask questions if you are unsure

• Always explain in your HW any unconventional action
  – worst result then is some points lost
  – worst result otherwise is expulsion

• Violations are unfair to other students and yourself
Quizzes

• Will have ≈5 quizzes during the quarter
  – 20-30 minutes each
  – probably multiple choice / short answer questions
  – may take place during the lecture period
    • make sure that time slot is available
  – details still TBD...
Books

**Required** book

- *Effective Java* 3rd ed, Bloch (EJ)

**Optional** book

- *Pragmatic Programmer*, new 20\(^{th}\) anniversary (2\(^{nd}\)) edition, Hunt & Thomas (PP)

**Other** books

- *Program Development in Java*, Liskov & Guttag
  - would be the textbook if not from 2001
- *Core Java* Vol I, Horstmann
  - good reference on language & libraries
Books? In the 21st century?

• Why not just use Google, Stack Overflow, Reddit, Quora, …?
• Web-search good for
  – Finding the parameters of a Java API function
• (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:
  – Answers on the web are often quickly out of date
    • aim is to answer the question at the time when asked
  – “This incantation solved my problem”
    • give that to users without knowing how it works?
Readings

• Calendar will include book sections for you to read
  – EJ = required, PP = optional

• These are “real” books about software, approachable in 331
  – occasionally slight reach: accept the challenge

• Overlap only partially with lectures
  – books include lots of other useful information

• Readings are fair game for quizzes
  – want to make sure you do it
Exams

• No real exams

• Our final “exam”
  – demo your final HW solution to a TA
  – answer some questions about your experiences writing it
# Grading

Approximate weighting (subject to change):

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>65%</td>
<td>Homework</td>
</tr>
<tr>
<td>25%</td>
<td>Quizzes</td>
</tr>
<tr>
<td>10%</td>
<td>Final “exam”</td>
</tr>
</tbody>
</table>
CSE 331 can be challenging

- Past experience tells us CSE 331 is **hard**
  - not my intention to make it difficult!

- Big change to move
  - **from** programming by trial & error
    - technique that does not work for building large scale software
  - **to** programming by careful design, reasoning, and testing

- Programming itself can be hard
  - surprisingly difficult to specify, design, implement, test, debug, and maintain even a simple program
CSE 331 can be challenging

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

- Assignments will take more time than you think (start early)
  - even professionals routinely underestimate by 3x
  - these assignments will be a step up in difficulty

- If you are having trouble, think before you act
  - then, look for help
Other Advice

• Don’t be afraid to make mistakes
  – accepting that you will make mistakes is perhaps the most important lesson of this course
  – we often learn best from our mistakes
  – if you’re not making mistakes, you’re not challenging yourself

• Don’t expect everything to be spelled out for you
  – real-world problems don’t come that way
    • if there are detailed instructions for solving a problem, then there should already be a program that does it
  – world needs you for your intuition, creativity, & intelligence
Problems
A Problem

“Complete this method such that it returns the location of the largest value in the first $n$ elements of the array $arr$.”

```c
int maxLoc(int[] arr, int n) {
    ...
}
```
Solution 1

```c
int maxLoc(int[] arr, int n) {
    int maxIndex = 0;
    int maxValue = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > maxValue) {
            maxIndex = i;
            maxValue = arr[i];
        }
    }
    return maxIndex;
}
```
Solution 2

```java
int maxLoc(int[] arr, int n) {
    int maxIndex = -1;
    int maxValue = Integer.MIN_VALUE;
    for (int i = 0; i < n; i++) {
        if (arr[i] > maxValue) {
            maxIndex = i;
            maxValue = arr[i];
        }
    }
    return maxIndex;
}
```
A Problem

“Complete this method such that it returns the location of the largest value in the first $n$ elements of the array $arr$.”

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

What questions do you have about the specification?
- what if $n = 0$?
- what if $n < 0$?
- what if $n > arr.length$?
- what if there are two maximum elements?
A Problem

“Complete this method such that it returns the location of the largest value in the first $n$ elements of the array $arr$.”

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

Could we write a specification with only one correct solution?
- throw IllegalArgumentException if $n \leq 0$
- throw ArrayIndexOutOfBoundsException if $n > arr.length$
- return smallest index achieving maximum
Morals

• You can all write the code

• Writing the specification was harder than the code
  – multiple choices for the “right” specification
    • have to carefully think through corner cases
  – once the specification is chosen, code is straightforward
  – (both of those will be recurrent themes)

• Some math (e.g. “if n <= 0”) often shows up in specifications
  – English (“if n is less or equal to than 0”) is often worse
An exercise before next class

• Do HW0 (90 minutes max) before lecture on Wednesday
  – write an algorithm to rearrange array elements as described
  – argue in concise, convincing English that it is correct
    • don’t just explain *what the code does!*
  – should run in O(n) time
    • (optional challenge: can you do it in a single pass?)
  – do not actually run your code!

• Start trying to *reason* about the code you write
  – this may be difficult... if so, remember that!
  – next, we will learn to use a set of tools that will make this easy
Before next class...

1. Familiarize yourself with website:

   http://courses.cs.washington.edu/courses/cse331/20sp/
   - read the syllabus
   - read the academic integrity policy
   - find the homework list
   - find the link to Canvas

2. Do HW0 before lecture on Wednesday!
   - limit this to 90 minutes
   - submit a PDF on Gradescope (invite coming today)
   - not graded