What is the goal of CSE 331?

In short: to help you become better programmers

Specifically, to teach you how to write code of
- higher quality
- increased complexity

We will discuss tools and techniques to help with these
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?

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

1. **Tools**
   - e.g., type checking compiler

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%
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 defect OS could sink the *whole fleet*

• And more reasons we will see shortly…
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

In summary, we want our code to be:

1. correct
2. easy to change
3. easy to understand
4. modular
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% are errors are btw modules [Seaman, ‘08]
  – corner cases are more important with more users
• Comm. costs dominate schedules [Brooks, ‘75]

**Corollary**: quality must be even higher, per line, in order to achieve overall quality in a large program
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

<table>
<thead>
<tr>
<th>Correctness</th>
<th>Changeability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Tools</td>
<td></td>
</tr>
<tr>
<td>• Git, Eclipse, JUnit, Javadoc, …</td>
<td></td>
</tr>
<tr>
<td>• Java libraries: equality &amp; hashing</td>
<td></td>
</tr>
<tr>
<td>• Adv. Java: generics, assertions, …</td>
<td></td>
</tr>
<tr>
<td>• debugging</td>
<td></td>
</tr>
<tr>
<td>2. Inspection</td>
<td></td>
</tr>
<tr>
<td>• reasoning about code</td>
<td></td>
</tr>
<tr>
<td>• specifications</td>
<td></td>
</tr>
<tr>
<td>3. Testing</td>
<td></td>
</tr>
<tr>
<td>• test design</td>
<td></td>
</tr>
<tr>
<td>• coverage</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Understandability</th>
</tr>
</thead>
<tbody>
<tr>
<td>• specifications, ADTs</td>
</tr>
<tr>
<td>• listeners &amp; callbacks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Changeability</th>
<th>Modularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>• specifications, ADTs</td>
<td></td>
</tr>
<tr>
<td>• listeners &amp; callbacks</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Understandability</th>
<th>Modularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>• specifications, ADTs</td>
<td></td>
</tr>
<tr>
<td>• Adv. Java: exceptions</td>
<td></td>
</tr>
<tr>
<td>• subtypes</td>
<td></td>
</tr>
<tr>
<td>• module design &amp; design patterns</td>
<td></td>
</tr>
<tr>
<td>• event-driven programming, MVC, GUIs</td>
<td></td>
</tr>
</tbody>
</table>
Administrivia
Course staff

• Lecturer:
  – Kevin Zatloukal (kevinz@cs, zat@uw)

• TAs:
  – Justin Bare (jbare@cs)
  – Vincent Liew (vliew@cs)

• Office Hours:

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:30 – 3:30pm</td>
<td>2:30 – 3:30pm</td>
<td>2:30 – 3:30pm</td>
</tr>
<tr>
<td>CSE 218</td>
<td>CSE 006</td>
<td>CSE 006</td>
</tr>
<tr>
<td>Kevin</td>
<td>Vincent</td>
<td>Justin</td>
</tr>
</tbody>
</table>
Staying in touch

• Course email list: cse331a_su16@u.washington.edu
  – for class announcements
  – students and staff already subscribed
  – fairly low traffic

• Message Board
  – for class discussion (staff will monitor and participate)
  – 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
Prerequisites

Only prerequisite is Java knowledge
  – we assume you have mastered CSE142 and CSE143

Examples

• Sharing:
  – distinction between \texttt{==} and \texttt{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 \texttt{extends} (classes) and \texttt{implements} (interfaces)
Lecture and section

• Both are required

• All materials posted, but they are visual aids
  – arrive punctually and pay attention
  – if doing so doesn’t save you time, one of us is messing up (!)

• Section will often be more tools- and homework-focused
  – especially next week: preparing for projects

• Will post other handouts related to class material on web site
  http://courses.cs.washington.edu/courses/cse331/16su/
Homework

• Homework assignments will
  1. give you more practice
  2. require you to apply the techniques learned in class
     • Pro Tip: think about which techniques are intended

• Four (4) late days for the quarter: save for emergencies
  – max 2 per homework, save them for later
  – email staff if you need to use 2 (may have started grading)

• We will have 10 homework assignments
  – first 3 are on paper, then all coding
  – early assignments come faster in summer quarter…
Homework (cont.)

Not as bad as it looks on the calendar…

<table>
<thead>
<tr>
<th>June</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monday</strong></td>
<td>13:10–14:10 Lecture EEB 037 <em>Course overview &amp; introduction</em></td>
<td>14:30–15:30 Office hours (vlew) CSE 006</td>
<td>13:00 <strong>HW0 due</strong></td>
<td>13:10–14:10 Lecture EEB 037 <em>Reasoning about straight-line code</em></td>
<td>13:10–14:10 Lecture EEB 037 <em>Reasoning about loops</em></td>
</tr>
<tr>
<td></td>
<td>14:30–15:30 Office hours (kevinz) Location: TBD</td>
<td></td>
<td></td>
<td>14:30–15:30 Office hours (jbare) CSE 006</td>
<td>17:00 HW1 due</td>
</tr>
<tr>
<td><strong>Tuesday</strong></td>
<td>13:10–14:10 Lecture EEB 037 <em>Writing loops</em></td>
<td>14:30–15:30 Office hours (vlew) CSE 006</td>
<td>13:10–14:10 Lecture EEB 037 <em>Specifications (pt 1)</em></td>
<td>13:10–14:10 Section EEB 037 Git &amp; Java tools &amp; HW3</td>
<td>17:00 HW3 due</td>
</tr>
<tr>
<td></td>
<td>14:30–15:30 Office hours (kevinz) Location: TBD</td>
<td></td>
<td></td>
<td>14:30–15:30 Office hours (jbare) CSE 006</td>
<td></td>
</tr>
<tr>
<td></td>
<td>23:59 <strong>Quiz 1 due</strong></td>
<td></td>
<td>23:00 <strong>HW2 due</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Academic Integrity

"The code you write must be your own."

- Read the course policy carefully
  - collaboration is encouraged, but…
  - do not share your HW code with others

- When in doubt, document your collaboration in your HW
  - at worst, you will lose a few points

- Cheating disrespects your colleagues and yourself
Books

**Required textbook**

- *Effective Java 2nd ed*, Bloch (EJ)

**Other useful books:**

- *Pragmatic Programmer*, Hunt & Thomas (PP)
  - **recommended** (usually required)

- *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 articles can
  – be out of date (without any indication this is so)
    • even 2014 is like 1960 in Internet years
  – rely on context that is not apparent on that page

• Books usually give better presentation of high level ideas
  – the purpose of a language feature or library
  – key reasons for its design

• Do use the Java 8 APIs (link on web site)
Readings & Quizzes

• We will have readings from first 2 (or 3) books
  – if not in EJ, then photocopies will be provided in class
  – these books are also on reserve at the library

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

• Quizzes to make sure you don’t skip the readings
  – short: 2-6 questions, usually multiple choice
  – take as many times as you want
Exams

• Midterm in class on Friday, July 22nd

• Final in class on Friday, August 19th

• Exams will be
  – focused on concepts learned in class
  – shorter than in normal quarters (1 hour each)
Grading

Approximate weighting (subject to change):

<table>
<thead>
<tr>
<th>Weight (%)</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>Homework</td>
</tr>
<tr>
<td>5%</td>
<td>Homework readability review</td>
</tr>
<tr>
<td>5%</td>
<td>Reading quizzes</td>
</tr>
<tr>
<td>20%</td>
<td>Midterm exam</td>
</tr>
<tr>
<td>20%</td>
<td>Final exam</td>
</tr>
</tbody>
</table>

Readability review: make sure your code is **understandable**

- ungraded readability review on either HW 5 or HW 6
- graded readability review on either HW 7 or HW 8 or HW 9
Acknowledgments

- Course designed/created/evolved/edited by others
  - Michael D. Ernst
  - Dan Grossman
  - David Notkin
  - Hal Perkins
  - Zach Tatlock (newcomer last quarter)
  - A couple dozen amazing TAs

- Hoping my own perspective offers benefits

- [Because you are unlikely to care, I won’t carefully attribute authorship of course materials]
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 brute-force, trial & error
  - to programming by careful design, reasoning, and testing

- 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

- Learning to program well is worth the effort
  - create solely with the power of your imagination
  - create software that positively affects the lives of many people
Questions?
Reasoning about code
"Complete this method such that it returns the index of the max of the first $n$ elements of the array arr."

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

Take a minute to think about how you’d write this…
A Solution?

Is this solution **correct**?

```java
int indexOfMaximum(int[] arr, int n) {
    int maxValue = arr[0];
    int maxIndex = 0;
    for (int i = 1; i < n; i++) {
        if (arr[i] > maxValue) {
            maxValue = arr[i];
            maxIndex = i;
        }
    }
    return maxIndex;
}
```
A Solution?

Is this solution **correct**?

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

**Corner cases:**
- What if there are ties?
- What if \( n \) is 0?

**Error cases:**
- What if \( \text{arr.length} < n \)?
- What if \( \text{arr} \) is null?
Morals

• You can all write the code!

• Takes work to show that the code is correct
  • **Step 1**: what does it mean to be correct?
    – that is called the “specification” for the function
    – can’t argue correctness if we don’t know what is correct

• Specifications are hard to write
  – there can be many corner cases and error cases
  – do we even want to specify behavior for all of these?
    • depends on the situation
    • will discuss stronger vs weaker specs next time…
You have homework!

• Homework 0, due in dropbox by 1pm Wednesday
  – **write** an algorithm to rearrange array elements as described
  – **argue** in concise, convincing English that it is correct!
  – should run in $O(n)$ time
    • challenge: can you do it in a single pass?
  – do not actually run your code!

• Start learning to reason about the code you write
  – this is the one homework that is *intentionally* difficult
  – stop after 2 hours (write up what you tried)
    • this HW grade is for participation not results
  – this will be easy in a week or so
To-Do List

Before the next class…

1. Familiarize yourself with website:
   
   http://courses.cs.washington.edu/courses/cse331/16su/
   
   – read the syllabus (esp. the advice section)
   – read the academic integrity policy
   – find the homework list

2. Do HW0 by 1 pm Wednesday!
   
   – limit this to 2 hours
   – submit a PDF into the dropbox