Can Great Programmers Be Taught?  
Experiences with a Software Design Class  

John Ousterhout  
Stanford University
Q: What is the most important idea in Computer Science?
A: Problem decomposition
... no-one teaches it

Elite programmers are >10x more productive
... no-one teaches elite skills
Teaching Great Programmers

Is it possible?

By whom?

How?
## CS 190: Software Design Studio

- **Iterative approach:** like English writing class:
  - Write
  - Get feedback
  - Rewrite

- **Small class:** ≤ 20 students

<table>
<thead>
<tr>
<th></th>
<th>1. Build large system from scratch</th>
<th>2. Revise based on code reviews</th>
<th>3. Add new features to another team’s code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Code reviews</td>
<td>Code reviews</td>
<td>Code reviews</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instructor reads ~20k lines of code
What are the Secrets?

● A few (somewhat vague) overall concepts
  ▪ “Working” isn’t good enough: must minimize complexity
  ▪ Complexity comes from dependencies and obscurity
  ▪ Strategic vs. tactical programming
  ▪ Classes should be thick
  ▪ Generic classes are simpler
  ▪ New layer, new abstraction
  ▪ Comments should describe things that are not obvious from the code
  ▪ Define errors out of existence
  ▪ The Martyr Principle

● Most constructive in the context of code reviews

● Course is more about red flags than recipes
Classes Should be Thick

Interface: everything that must be known to users

Class

Useful functionality provided by class

Thin Class

Thick Class

Reformulation of classic Parnas paper: “On the Criteria to be Used in Decomposing Systems into Modules”
private void addNullValueForAttribute(String attribute) {
    data.put(attribute, null);
}
Classes Should be Thick, cont’d

- Common wisdom: “classes and methods should be short”
- Result: classitis
- Rampant in Java world:
  ```java
  FileInputStream fileStream =
      new FileInputStream(fileName);
  BufferedInputStream bufferedStream =
      new BufferedInputStream(fileStream);
  ObjectInputStream objectStream =
      new ObjectInputStream(bufferedStream);
  ```
- Length isn’t the big issue, it’s abstraction
A Thick Interface

- Unix file I/O:
  ```c
  int open(const char* path, int flags, mode_t permissions);
  int close(int fd);
  ssize_t read(int fd, void* buffer, size_t count);
  ssize_t write(int fd, const void* buffer, size_t count);
  off_t lseek(int fd, off_t offset, int referencePosition);
  ```

- Hidden below the interface:
  - On-disk representation, disk block allocation
  - Directory management, path lookup
  - Permission management
  - Disk scheduling
  - Block caching
  - Device independence
Define Errors Out of Existence

- Exceptions: a huge source of complexity
- Common wisdom: detect and throw as many errors as possible
- Better approach: define semantics to eliminate exceptions
- Example mistakes:
  - Tcl `unset` command
    (throws exception if variable doesn’t exist)
  - Windows: can’t delete file if open

Overall goal: minimize the number of places where exceptions must be handled
Is the Course Working?

- Hard to know: ask students in 5-10 years?
- Just the first step towards becoming a great programmer
- Good energy in class:
  - Tone of discussions changes halfway through
  - Students are thinking about their code in new ways
- Interesting challenges for me:
  - What causes complexity?
  - How to design simple code?
- Discovering new ideas from reading students’ code
  - Specialized → complicated
  - Generic → simple
Software Design Book

- My sabbatical project: capture ideas from CS190
  - Reach more people
  - Start a discussion
  - Define terminology

- Relatively short (~120 pages)

- Status:
  - First draft complete
  - About to get first round of reviews & comments
  - Self-publish by end of 2017?

Will the design ideas make sense standalone, without code reviews?
Conclusion

- It is possible to teach software design
  - But not currently scalable
- Principles gradually emerging
- Need more experience, input
Questions/Discussion