Software Design

John Ousterhout
Stanford University

Software Design

- **Working code isn't enough**
- **Code structure also important**
- **Goals for today:**
  - Convince you that software design matters
  - Give you a few ideas to think about
  - Inspire you to become great software designers

Software can be beautiful, and creating it is fun

I Love to Code!

- **Programming: purest creative medium in the history of the world?**
- **Not limited by physical constraints**
- **If you can imagine it, you can build it**
- **Limitations are intellectual: can we understand what we have built?**

The Enemy: Complexity

- **Over time, software systems become larger, more complex**
- **Complexity is the fundamental limit in software development:**
  - Hard to implement new functionality
  - Fixing one bug introduces another
- **No-one can keep the entire system in their head**
- **Have you gone back to code you wrote a while ago and had trouble understanding it?**

How can we design software to minimize complexity?

What is Complexity?

“Anything that makes it hard to understand and modify a software system”

- **Dependencies**
  - Can’t make a change here without understanding code elsewhere
  - If one piece of code changes, other pieces must change also
  - Must be able to make progress without understanding entire system!
- **Obscurity**
  - Important information is not obvious
    - What a variable is used for
    - How a piece of code works
    - Reasons why code has to be written a certain way
    - Dependencies

Hard To Appreciate?

- **Hard to discuss software design in CS 61B:**
  - Not enough experience with large systems
  - Haven’t seen consequences of bad design
- **Goal for today: plant a few seeds**
  - May not completely make sense right now

Managing Complexity

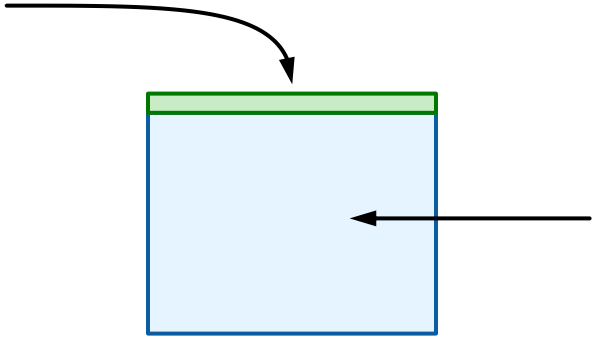
- **Eliminate it**
  - Eliminate special cases
  - Good names
  - Helpful comments
- **Hide it**
  - Developers don’t need to be aware of most of it most of the time
  - Modular design

Hiding Complexity

- **Large systems will inevitably have a lot of complexity**
- **What really matters is apparent complexity:**
  - How much code must developer write to complete current task?
  - How much knowledge does developer need to write the code?
  - How hard is it to get that knowledge?
- **Solution: modular design**
  - Divide system into relatively independent modules

Module

Interface:
everything someone needs to know in order to use module (cost)



Implementation:
code that implements useful functionality (benefit)

Modules exist at many levels:

- **Method**
- **Class**
- **Subsystem**
- **Application**

Interface

Everything someone needs to know in order to use a module

- **Example: class**
  - Method signatures (parameter names and types, result types)
  - Public variables (avoid if possible...)
  - Meaning of results
  - Constraints:
    - Legal values for parameters
    - Ordering requirements for methods
  - Side effects: ways in which methods impact the future behavior of the system
    - Modifications to files or internal state
  - Performance implications
  - ...

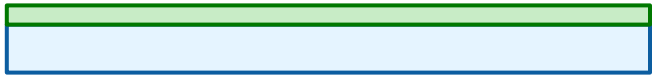
Part of code

Not captured by code

Interfaces, cont'd

- **Leverage against complexity:**
  - Learn something simple (interface)
  - Get lots of power
- **Abstraction: a simplified view of an entity, which omits unimportant details**
- **The best modules are deep: simple interfaces, lots of functionality**

Deep Module



Shallow Module

Deep Interfaces

Spreadsheet

	A	B	C	D
1				
2		18		
3		243		
4		990		
5		4		
6		67		
7		133		
8		8		
9		0		
10		411		
11		1874		
12				
13				
14				
15				
16				

Unix/Linux file system

```
int open(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, void* buffer,
              size_t count);
off_t lseek(int fd, off_t offset,
            int referencePos);
```

Implementation has 100's of Klines

Shallow Modules

- **Don’t have much functionality**
  - Interface looks a lot like the implementation
- **Complex interfaces:**
  - Many methods
  - Methods have many arguments
  - Complex behaviors and/or side effects
- **Shallow method:**

```
private void addNullValueForAttribute(String attribute) {
    data.put(attribute, null);
}
```

- **Shallow class from CS 61B:**
  - Graph class from Project 4

More Thoughts on Interfaces

- **Interfaces are good, but more interfaces are not necessarily better!**
- **General-purpose interfaces are deeper than specialized ones**
- **Interfaces require comments!**
  - Method signatures don't provide all the information needed to use a class or method
  - Must use comments to describe the rest of the interface
- **No comments?**
  - Users must read the implementation of the module
  - Even then, hard to synthesize the abstraction (e.g. overall rules of behavior)
  - Example: Project 4 from CS 61B

Managing Complexity

- **Eliminate it**
  - Eliminate special cases
  - Good names
  - Helpful comments
- **Hide it**
  - Developers don’t need to be aware of most of it most of the time
  - Modular design

Obvious Code

- **Code is obvious if:**
  - Someone can read it quickly and feel confident that they understand it
  - They can make guesses about its behavior without much thought
  - Those guesses will be right!
- **Things that make code non-obvious:**
  - Dependencies
  - Special cases (lots of `if` statements?)
  - Deep nesting
  - Missing comments (e.g., reasons why code has to be this way)
  - Bad names

Names

- **Good names make code more obvious**
- **Test: if someone sees this name in isolation, will they be able to guess its meaning?**
- **Biggest problem with names: too vague**

```
private Graph graph;
Queue<Integer> queue;
boolean blinkStatus;   boolean cursorVisible;
```

- **Nonobvious names from Project 4:**

```
findHyponyms              (single root hyponyms)
findMoreHyponyms          (multiple hyponyms find intersection)
popularWords              (popular hyponyms from findMoreHyponyms result)
```

Names, cont'd

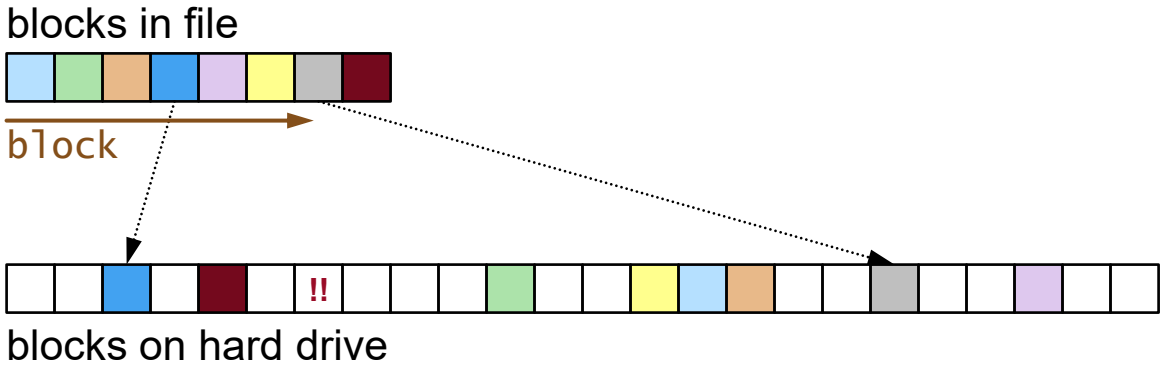
- **Be consistent!**
  - Using a particular kind of thing in many places? Always use the same name.
  - Don't use the same name for different things
- **Very short names sometimes OK**

  ```
  for (i = 0; i < num_hyponyms; i++) {...}
  ```

  (but never use `i` and `j` for anything other than loop indices)
- **Longer names aren't always more obvious**

  ```
  isLeastRelevantMultipleOfLargerPrimeFactor
  ```
- **If you're having trouble finding a clear and simple name, that's a red flag:**
  - Entity has complex meaning
  - Change the meaning?
  - Divide into two simpler things?
  - Merge with something else?

The Worst Bug



- **Blocks in files suddenly turned to all zeroes**
- **Took 6 months to track down**
- **Cause: name `block` used for different purposes:**
  - Logical block number in file
  - Physical block number on hard drive

Other Thoughts on Design

- **Must decide what's important, focus on that**
  - Make the most common activities simple and easy
  - Capture the essence of a variable in its name
- **Can't design entire system up-front**
  - Can't visualize consequences of choices
  - Make best guess, refactor like crazy
- **Complexity is incremental:**
  - Accumulates in tiny chunks
  - Must sweat the small stuff!

Becoming a Great Software Designer

- **Must teach yourself**
  - No courses
  - Even in industry, may not get much help
- **Get and give feedback**
- **Challenge everything:**
  - Why does this code seem to be complex? How could it be made more obvious?
  - Is this the simplest possible API for this class?
- **Learn from your mistakes**
  - You won't get it right the first time (no-one does)
  - Take time to fix problems
- **Vote with your feet:**
  - Work for companies that care about design

What About AI??

- **Developers may spend more time on design**
  - AI tools will fill in the details
- **Will AI tools take over design?**
- **Tools can compensate for complexity**
  - Explain code
    - Where is a particular action taken?
    - Compensate for lack of comments
  - Identify dependencies (must know enough to ask)
- **AI tools make great collaborators**
  - Source of suggestions, critiques
  - Can find some bugs
  - Help you become a great designer sooner?

Conclusion

- **Software design is important**
  - Complexity limits our ability to create powerful systems
- **Software design is fun**
  - Fascinating challenges
  - Great feeling when successful

Start thinking about more than just whether your code works