CS1530, Lecture 18: Software Craftsmanship

Bill Laboon

Software Engineer != (Assembly Line Worker || Salesperson || Soldier)

- Hard to quantify what you do
- Not repetitive – almost by definition, every challenge is different
- Often your manager will not understand what you are doing
- Being great is subjective and UP TO YOU

The Three Virtues of a Great Programmer (Larry Wall)

- Laziness
- Impatience
- Hubris

Laziness

- A great programmer will take pains to reduce OVERALL work
- Write labor-saving programs that they and other will find useful
- Programs have few defects, are understandable and well-documented, to avoid having to answer questions and support it

Impatience

- Great programmers are angry when the computer is lazy and inefficient
- It's just sitting there wasting cycles! Those cycles could be helping me do my work (automation)
- It takes too long for the computer to understand what I'm doing – I should come up with better ways for it to react to my needs – or even anticipate them

Hubris

- Great programmers want to write great programs. You have to believe you can achieve greatness before you can try to do so.
- These programs shall be sung about by bards in the distant future!
- Note: The Billiad and the Laboonyssey have not yet been written about me…

All that said…

- An important aspect of software engineering is also *intellectual humility*
- Writing software is HARD.
- Our computers get more capable, but they are continuously pushed to their limits by ever-increasingly complex software

Intellectual Humility

"The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague."

-Edsgar Dikjstra, "The Humble Programmer"
https://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html

Intellectual Humility

"[O]ne programmer places a one-line program on the desk of another and either he proudly tells what it does and adds the question "Can you code this in less symbols?" —as if this were of any conceptual relevance!— or he just asks "Guess what it does!".

... I am sorry, but I must regard this as one of the most damning things that can be said about a programming language."

-Edsgar Dikjstra, "The Humble Programmer"
https://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html

Intellectual Humility

"Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?"

-Brian Kernighan, "The Elements of Programming Style"

What is Intellectual Humility?

- It is intellectual *honesty*
- Don't pretend to be an expert if you are not – even if you are, understand your weak points
  - γνῶθι σεαυτόν – Know thyself!
- Readily admit your mistakes, and be prepared to throw away your work if you did it incorrectly

What is Intellectual Humility?

- Be realistic in your status reports and estimates
  - UPOD – Underpromise and overdeliver
  - Estimation is hard! But programmers tend to be better at it than managers.
  - Do not be afraid to “manage up”

Reduce Complexity

```
// Let's imagine a simple Java program:

public class Foo2 {
    public static void main(String[] args) {
        int i = 0;

        for (int j = 0; j < 10; j++) {
            i = j * 2;
            System.out.print(j + " doubled is ");
            System.out.println(i);
        }
    }
}
```

Reduce Complexity

```
$ java Foo2
0 doubled is 0
1 doubled is 2
2 doubled is 4
3 doubled is 6
4 doubled is 8
5 doubled is 10
6 doubled is 12
7 doubled is 14
8 doubled is 16
9 doubled is 18
```

Reduce Complexity

- Here is the bytecode, produced via a complex parsing, compilation, and linking process for an imaginary machine (the “JVM”)…

Reduce Complexity

```
public class Foo2 {
  public Foo2();
    Code:
       0: aload_0
       1: invokespecial #1
       4: return


  public static void main(java.lang.String[]);
    Code:
       0: iconst_0
       1: istore_1
       2: iconst_0
       3: istore_2
       4: iload_2
       5: bipush        10
       7: if_icmpge     52
      10: iload_2
      11: iconst_2
      12: imul
      13: istore_1
      14: getstatic     #2
      17: new           #3
      20: dup
      21: invokespecial #4
      24: iload_2
      25: invokevirtual #5
      28: ldc           #6
      30: invokevirtual #7
      33: invokevirtual #8
      36: invokevirtual #9
      39: getstatic     #2
      42: iload_1
      43: invokevirtual #10
      46: iinc          2, 1
      49: goto          4
      52: return
}
```

Reduce Complexity

- Here is the actual assembly run on my machine (remember the JVM is *virtual* machine)…

Reduce Complexity

```
0x000000010708c760: mov %eax,-0x14000(%rsp)
0x000000010708c767: push %rbp
0x000000010708c768: sub $0x30,%rsp
0x000000010708c76c: movabs $0x11afbd17,%rax
; {metadata(method data for {method} {0x000000011adfdff8}
'hashCode' '()I' in 'java/lang/String')}
0x000000010708c776: mov 0xdc(%rax),%edi
0x000000010708c77c: add $0x8,%edi
0x000000010708c77f: mov %edi,0xdc(%rax)
0x000000010708c785: movabs $0x11adfdff8,%rax
; {metadata({method} {0x000000011adfdff8} 'hashCode' '()I' in
'java/lang/String')}
0x000000010708c78f: and $0x1ff8,%edi
0x000000010708c795: cmp $0x0,%edi
; ~1,900 lines after this..
```

Reduce Complexity

- All of the previous files are under `/sample_code/complexity` in the class repo if you'd like to explore further
- Point is, a computer is a massive stack of abstractions
- Even assembly is an abstraction on top of 1's and 0's, which is just an abstraction on top of high and low voltages…

Reduce Complexity

- Software Engineering is about making USEFUL abstractions
- What is useful?
  - Understandable
  - Hide complexity
  - ..while still making development possible!

How To Reduce Complexity

- Avoid global mutable state
  - Global variables make testing and understandability harder
  - Try to have "pure" functions when possible (functions who only return a result, and whose results depend only on arguments)
    - Extract these when possible
  - This will also help you with multi-threading programs

How To Reduce Complexity

- Deepness is your enemy
  - Avoid deep hierarchies and object ontologies
  - Avoid deep nesting of loops
  - Avoid too much "magic" – ideally, developers can understand/modify your abstractions if necessary
    - Example: Meta-programming in Ruby – I once redefined the keyword "loop" (think of a Java while) in a test
    - Do you think this helped others understand how the code worked?

How To Reduce Complexity

- Using a well-defined code style guide
  - Easier to understand at a glance – in Java, I know that NUM_KITTY_CATS is a constant without looking at its definition
  - Spend more time arguing algorithms and less time arguing over camelCase vs PascalCase vs Kebab-Case vs snake_case
- Use conventions, "convention over configuration" in Rails world
  - If your stack has a standard way of doing things, usually makes sense to do it that way
  - Only avoid it if you have a good reason

How To Reduce Complexity

- Routines/methods/functions should be short (< 10 lines in Java)
  - Use Wrapper and Sprout techniques to partition methods
  - Methods provide mental "entry and exit" points
  - GOTO is considered harmful for a reason (Edsger Dikjstra, "Go To Statement Considered Harmful" - http://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf)
  - COME FROM is even worse, but thankfully rarer

Paradox

- Should it concern you that some of these things are mutually exclusive?
- Yes! But it has the best our field has done.
- EVERYTHING in software development is trade-offs.
- ^- Key takeaway from this class.

Programming For People

- Write programs for people first, computers second
  - Engineering time is expensive, computer time (in general) is not (of course there are exceptions)
- Making code readable by humans means increased:
  - Comprehensibility
  - Reviewability
  - Fewer errors
  - Easier debugging
  - Less development time
  - Better external and internal quality
  - Easier to modify in the future

Don’t Fall For Gurus (Not Even Me)

- Read and understand others’ opinions, but…
- Make up your own mind.
- Someone offering you a silver bullet is offering you snake oil.

Welcome to a Dynamic Field

- Interested in working with mechanical watches?
- The last major change to how mechanical watches work was in 1930, with the Rolex Oyster Perpetual's improvement on Harwood's auto-winder
- There have been incremental improvements (and of course electronic/quartz watches are a whole other story), but nothing in a modern mechanical watch would surprise a watchmaker from, say, 1935

Software Engineering != Watchmaking

- Would anything in a modern computer surprise our hypothetical counterpart from 1935?

Zuse Z1

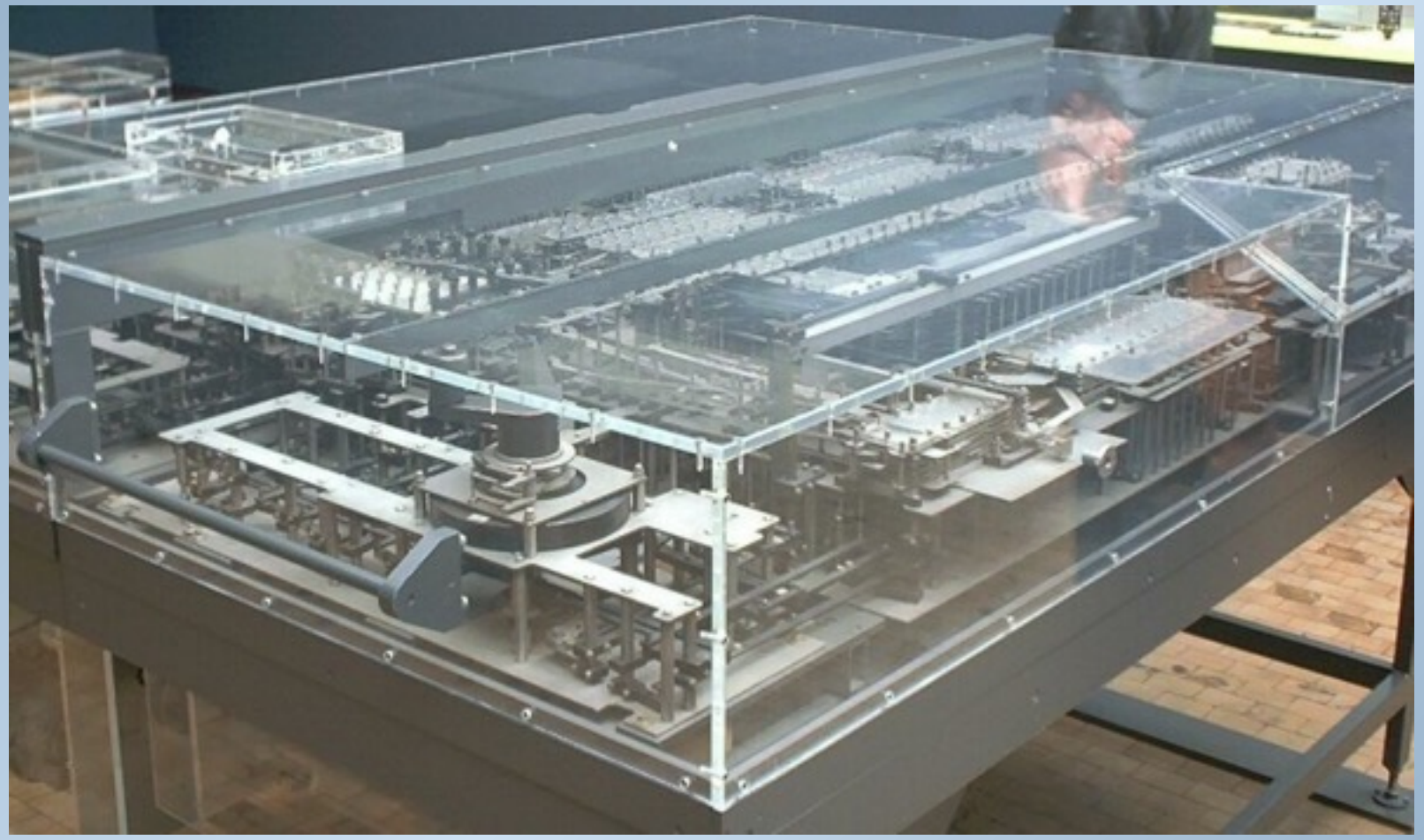

Image credits: https://en.wikipedia.org/wiki/Z1_(computer)#/media/File:Zuse_Z1.jpg

In some ways, yes, in others, no!

"Z1 was a machine of about 1000 kg weight, which consisted from some 20000 parts. It was a programmable computer, based on binary floating point numbers and a binary switching system. It consisted completely of thin metal sheets, which Kuno and his friends produced using a jigsaw."[6] "The [data] input device was a keyboard...The Z1's programs (Zuse called them Rechenplans) were stored on punch tapes by means of a 8-bit code."

– "**Konrad Zuse—the first relay computer" http://history-computer.com/ModernComputer/Relays/Zuse.html**

Staying Up-To-Date

"Mr. Morgan, what will the stock market do next year?"

"It will fluctuate, my boy, it will fluctuate."
-Attributed to J.P. Morgan in "The Intelligent Investor" by Benjamin Graham

Staying Up-To-Date

- Similarly, the only thing I can guarantee you in our field is that it will change
- Will functional programming take over the world? Will property-based testing become the new standard? Will Elixir be the web development language of choice?
- These are my opinions, but who knows… I have been wrong before and will be wrong again
- But I do know that those who don't change will find themselves left behind.

What Can I Do to Stay Up To Date?

- Read books
- Read Hacker News
- Learn new languages, framework, and libraries that sound interesting! Especially ones that are different from what you normally use…
  - Haskell, Lisp, Prolog, J, Ruby, Julia, Elixir, Erlang, Rust, Go, Dart, Racket, etc.

What Can I Do to Stay Up To Date?

- Meet with other programmers! Especially ones not at your company or on your team
- Give talks!
- Go to conferences
- Contribute to open source projects
- Constantly think, “could this be done better?”

As A Software Engineer, You Are A Professional

- Act as a professional
  - This doesn't mean a suit and tie, it means respect people and your work
- Professionals care about what they do more than whom they are working for
- Keep ethics in mind
  - Volkswagen Emission Scandal
  - THERAC-25
  - StuxNet