Welcome to CS 362

- Applied Software Engineering
- What happens after (and during) design?
- Testing, debugging, maintaining programs
  - Lessons for software testing
    - Methods for testing
    - Tips, tricks, and techniques
  - How to review code
  - How to debug
  - Throughout: tools and automation
    - Source control, testing, debugging tools

Overview

- This class is going to spend a lot of time on the fact that *things go wrong*
  - The design is probably somewhat wrong
  - The implementation is almost certainly wrong

  - Building software by “trying really hard” and assuming the end result is perfect *will not work*
  - There are lots of ways to avoid bugs
  - There are lots of ways to find bugs
  - There are lots of ways to fix bugs
  - There are lots of bugs

Overview

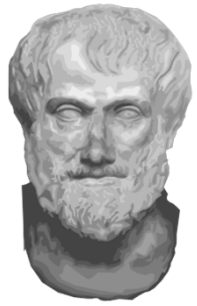
- Some books (not textbooks) that are worth reading to understand applied software engineering

- This class will aim to distill the most important points from a number of books into takeaway lessons
  - But the books are still worth reading if you really want to understand things

The Royal Road to Software Engineering Capability

- Build, test, and debug real software!
  - This is the most important thing

“he who learns to play the harp learns to play by playing it”
- Aristotle, *Metaphysics, Book IX*

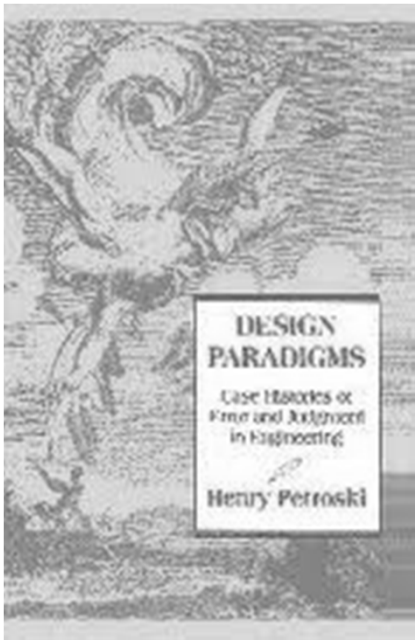


- Read books by people who have done just that, and written down what they have learned
  - If you think that’s too old fashioned, read the books on your tablet, ok?

Civil Engineering

- *Design Paradigms: Case Histories of Error and Judgment in Engineering*

  - Henry Petroski



- *"It has long been practically a truism among practicing engineers and designers that we learn much more from failures than from successes."*

Civil Engineering

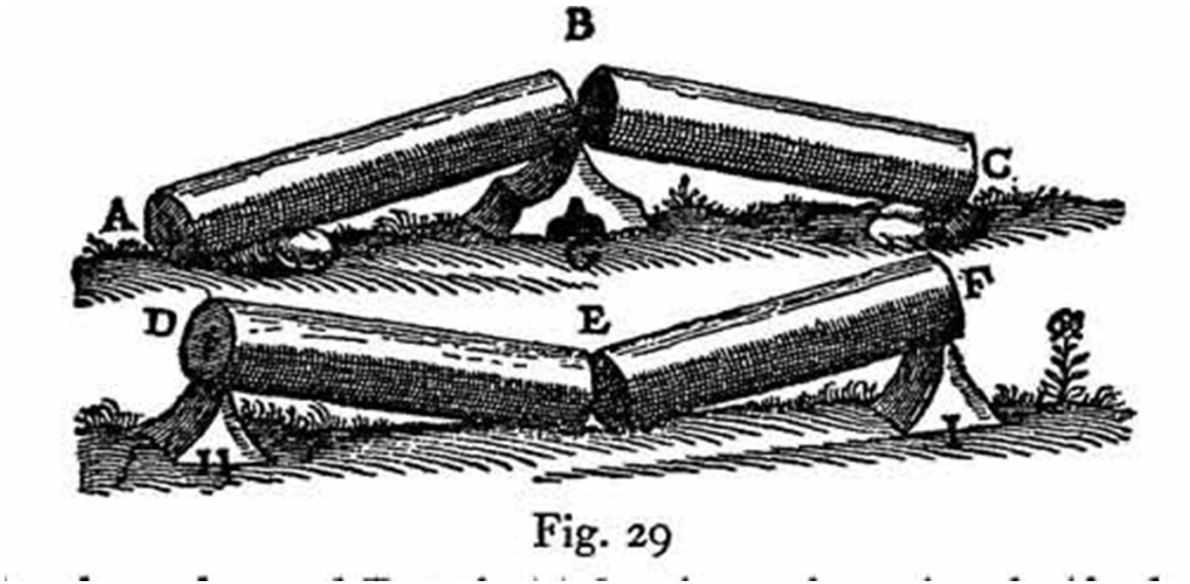


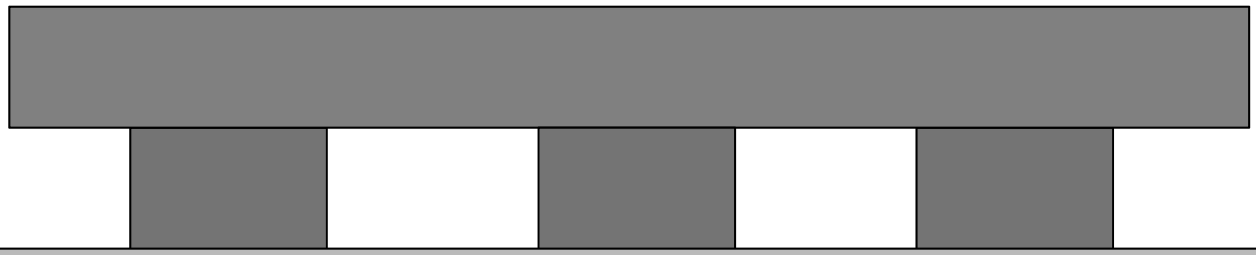
Fig. 29

Galileo's "bug report"

A "patch" to the code

Storing columns during construction:

A procedure "run" by the builders



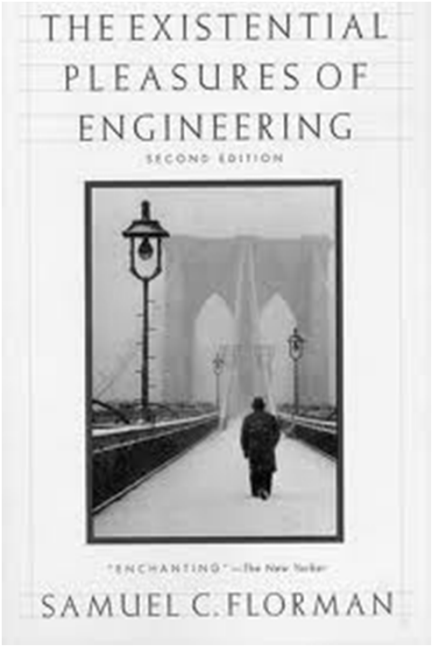
We're Engineers – Really!

- What makes software engineering *like* other engineering disciplines?
  - Design is *difficult and important*
  - Implementation is *difficult and important*
  - Building complex artifacts that must *work*
  - Use of mathematical principles, scientific understanding (e.g., materials science, physics / complexity theory, programming language knowledge)
  - Contributes to the material and cultural capital of humanity

Engineering and Humanity

- *The Existential Pleasures of Engineering*

  - Samuel C. Florman



- All engineers as engaged in one of the most profound and extraordinary quests of human existence

We're *A New Kind* of Engineers

- What makes software engineering *different* (in a bad way)?
  - We have over 2,500 years less experience
  - Computer science is simply less mature than civil or mechanical engineering
  - Many more moving parts!
  - Discrete behavior vs. continuous
    - Calculus and differential equations will not carry us very far
  - Many more radically different designs

We're *A New Kind* of Engineers

- What makes software engineering *different* (in a good way)?
  - Patching code is much easier than patching a bridge or a space shuttle
  - We know a lot about how to put computers to work to help us do our jobs
  - *We have much more ability to test*
  - Therefore testing will be the most important and most central topic of this class

Think-piece

You are given a library, container.o and the following .h file signatures:

```
int put (int n, container* c); /* returns 1 and adds to c if n not in c, otherwise returns 0 */
int get (int n, container* c); /* returns 1 if n is in c, 0 otherwise */
int remove (int n, container* c); /* returns 1 if n was in c; after return n is not in c! */
container* newContainer(); /* returns a new container if memory avail */
```

You don't have source code, and the file isn't compiled with debugging information. Attached is a note: "We would like to use this (it's really fast) in our new system, but it needs to work well – a bug in this could be catastrophic. Can you give me a plan/approach for thorough testing? I don't want to share our test generation code with the company that wrote this, and they won't share source, but you can give them test cases. The programmer behind this at Container Code Design, LLC, is pretty busy, so we'd like to make sure to get good turnaround from debugging. Can I get a short white paper on this by this afternoon's 2:35 project meeting? I know it's short notice, and you're not really a test engineer, but we need something.