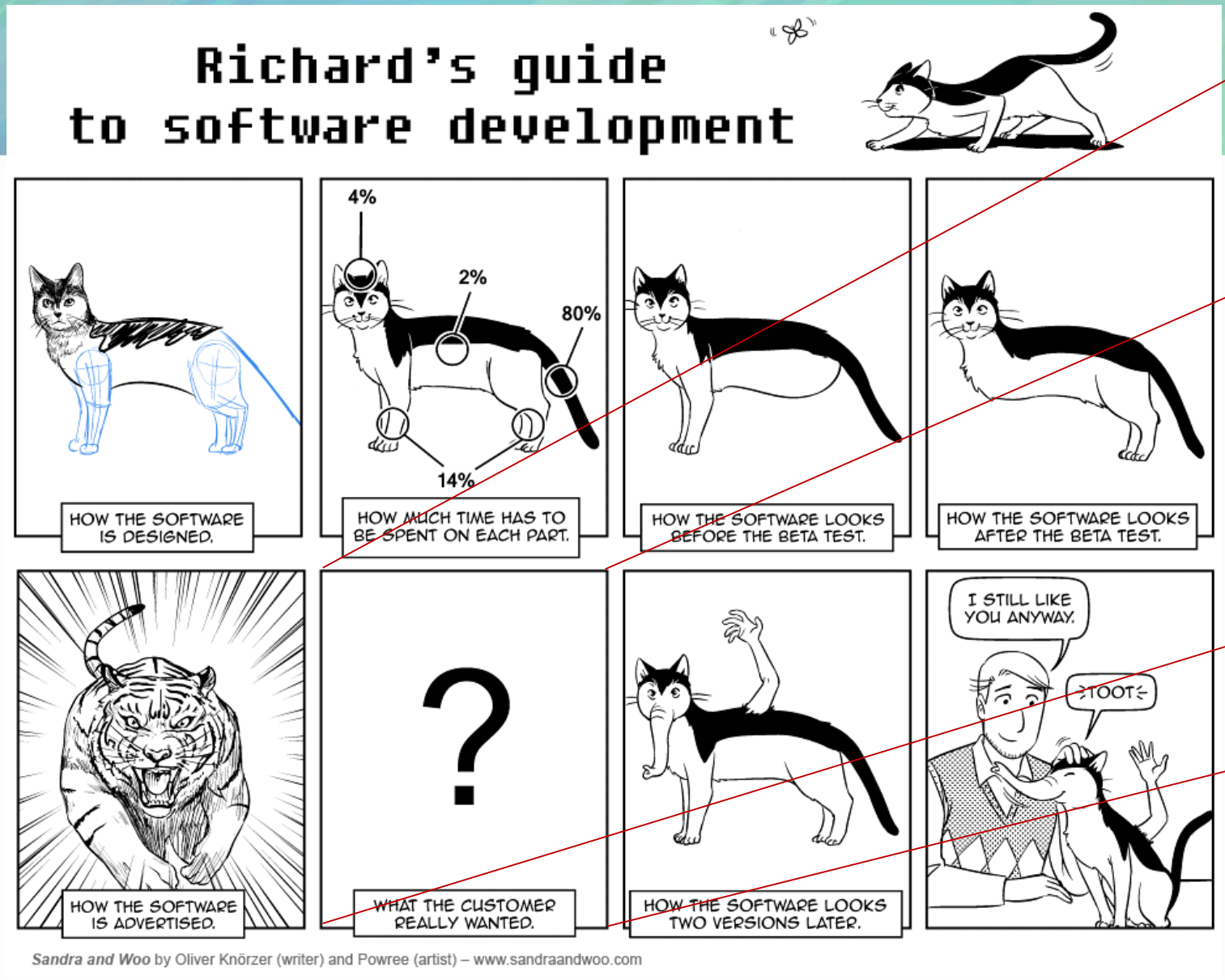
Introduction to Software Engineering

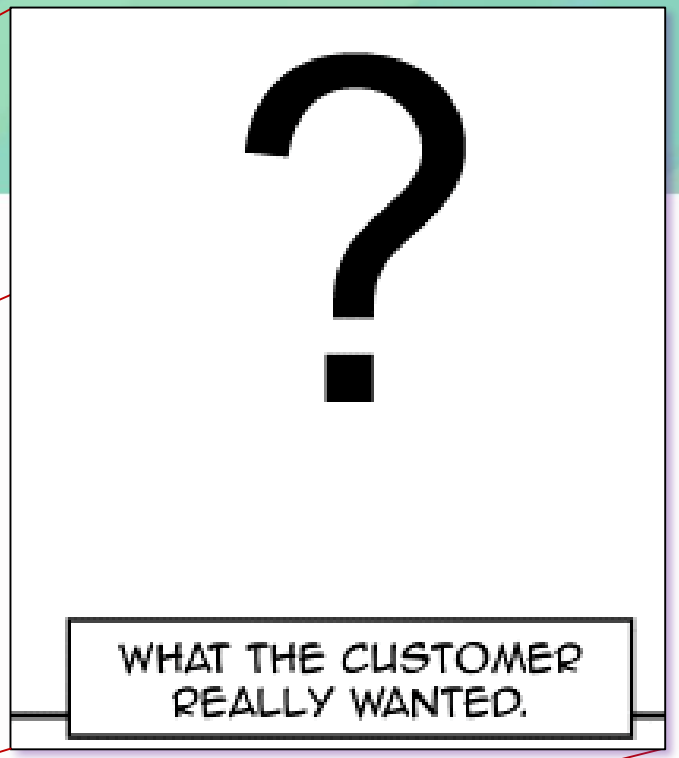
CS 2110 | Reading: MSD ~ Chapter 1

These slides supplement the textbooks

Announcements

- **First exam is over!** Yay! Grading is underway
- **Quiz**: out this Friday, due Sunday @ 11:30pm as usual
- **No In-Class Activity today** (there will be one on Friday, though!)
- **Please consider sharing your notes** - Still looking for note takers
- **Academic Integrity Policy Messages** - Simply be mindful of what you're doing, you reap the benefits by adhering to the integrity policy


Richard's guide
to software development
HOW THE SOFTWARE IS DESIGNED.
4%
2%
80%
14%
HOW MUCH TIME HAS TO BE SPENT ON EACH PART.
HOW THE SOFTWARE LOOKS BEFORE THE BETA TEST.
HOW THE SOFTWARE LOOKS AFTER THE BETA TEST.
HOW THE SOFTWARE IS ADVERTISED.
?
WHAT THE CUSTOMER REALLY WANTED.
HOW THE SOFTWARE LOOKS TWO VERSIONS LATER.
I STILL LIKE YOU ANYWAY.
≥TOOT≤
Sandra and Woo by Oliver Knörzer (writer) and Powree (artist) – www.sandraandwoo.com


?
WHAT THE CUSTOMER REALLY WANTED.

What's a big program?

- How many **classes** in your largest program?
  - 3 or fewer
  - 4-6 classes
  - 6-12 classes
  - 13-20 classes
  - more than 20 classes
- How many **lines of code** in your largest program?
  - 200 or fewer
  - 200 – 400
  - 400 - 1,000
  - 1,000 – 2,000
  - 2,000 or more

"KLOC" – Thousands of Lines of Code

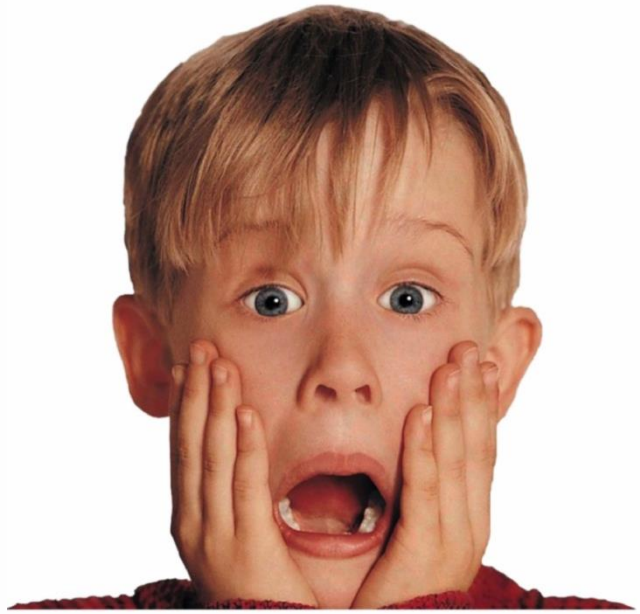
What's a big program?

Table from the textbook:

Category	Programmers	Duration	Product Size (Lines of Code)
Trivial	1	1 - 4 weeks	< 500
Small	1 - 2	1 - 6 months	500 - 3,000
Medium	2 - 5	6 mo - 2 yrs	3,000 - 20,000
Large	5 - 20	2 - 3 yrs	20,000 - 100,000
Very Large	20 - 200	3 - 6 yrs	100,000 - 1,000,000
Extremely Large	> 200	> 6 yrs	> 1,000,000

No Software Systems?

- Can you get through a day **without** interacting in any way with any software systems?
- Think about it… *can you?*
- Turn to your neighbor and discuss what you could *actually* do in your day… ☺



What's the Problem?

- Software costs are increasing as hardware costs decline.
- Many failures attributed to software
- Many software development disasters:
  - Cost overruns, late delivery
  - Reduced or wrong functionality, non-existent documentation
- Cost of failure becoming very high:
  - Inconvenience
  - Financial loss
  - Loss of life or loss of equipment

Software is becoming so **prevalent** in nearly everything we do – failures impact **everyone**

Software Failures or Disasters

- On 21 September 1997, a division by zero error on board the USS Yorktown (CG-48) *Remote Data Base Manager* brought down all the machines on the network, causing the ship's propulsion system to fail.
- The warship was testing its new Smart Ship system, which uses off-the-shelf PCs to automate tasks that sailors have traditionally done themselves.

Iowa: app blamed for slighting candidate

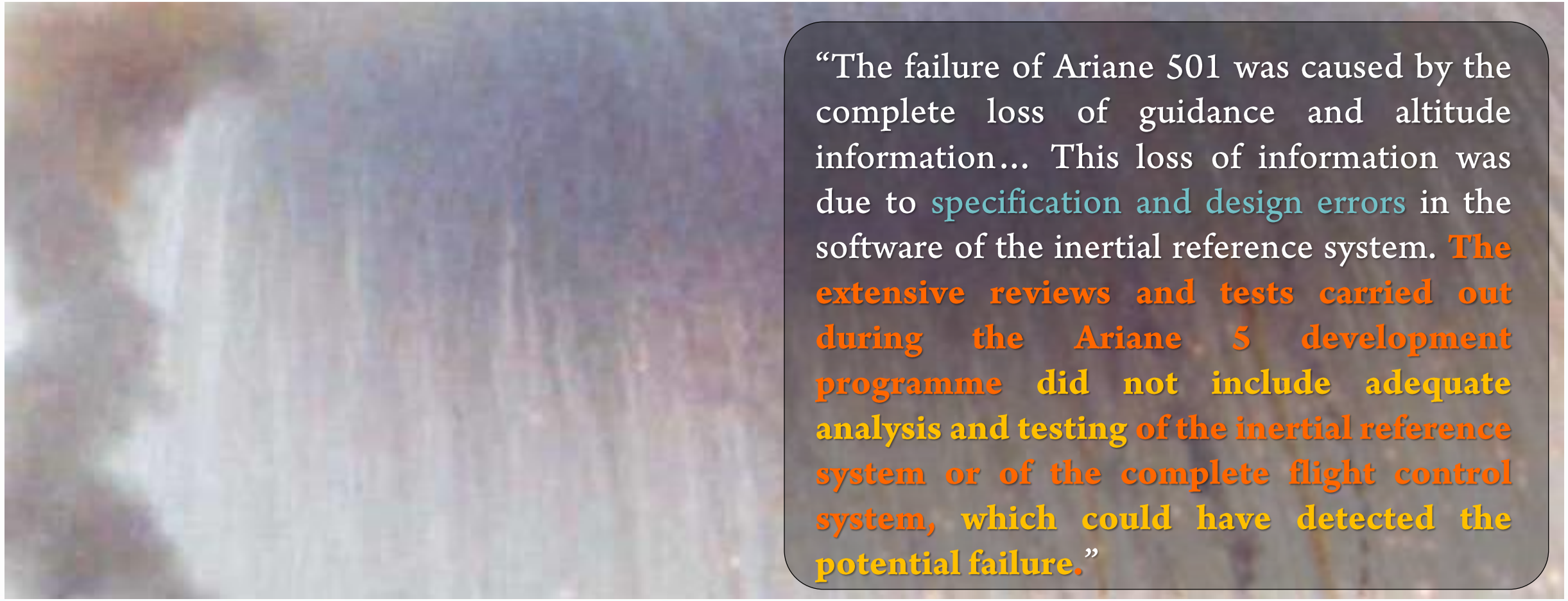
"Honestly, there is no need to attribute conspiracy or call shenanigans on what happened with the new app during the Iowa caucuses," Dan McFall, chief executive at app testing company Mobile Labs, told me in an email. "It's a tale that we have seen with our enterprise customers for years: A new application was pushed hard to a specific high profile deadline. Mobility is much harder than people realize, so initial release was likely delayed, and to make the deadline, they cut the process of comprehensive testing and then chaos ensues."

Twitter

Ariane 5 launch explosion, 4 June 1996. (http://www.esa.int/esaCP/Pr_33_1996_p_EN.html)

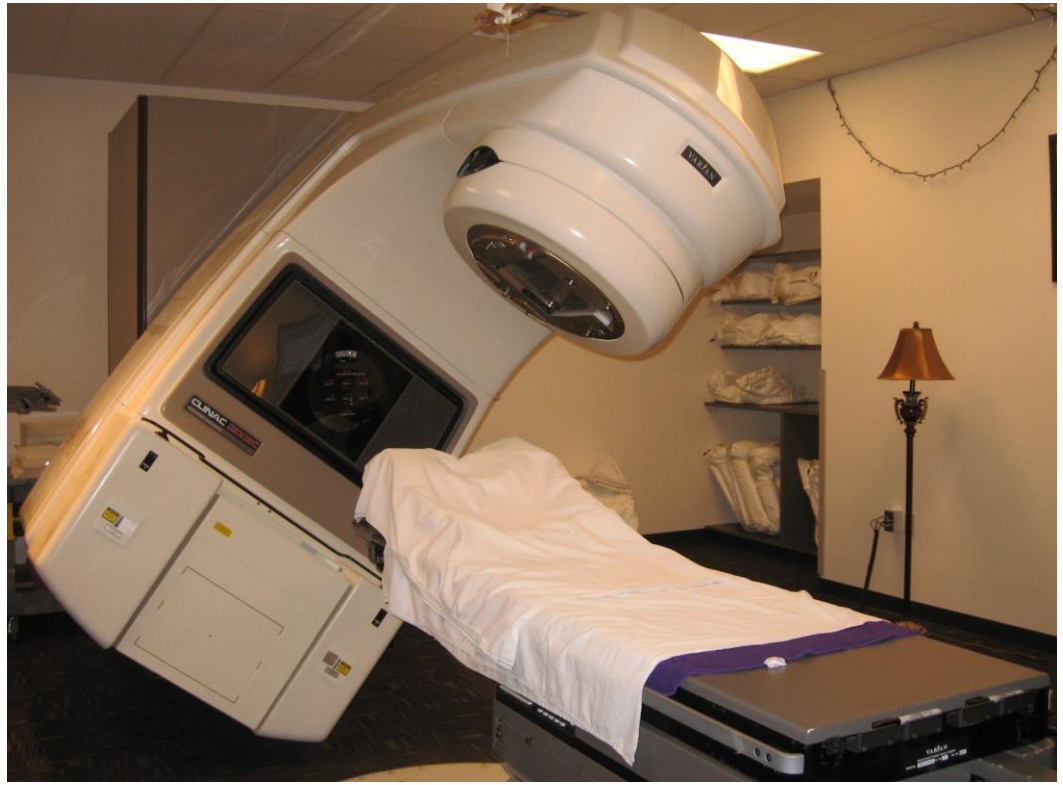
Software Failures or Disasters

Ariane 5: What we got …



"The failure of Ariane 501 was caused by the complete loss of guidance and altitude information… This loss of information was due to specification and design errors in the software of the inertial reference system. **The extensive reviews and tests carried out during the Ariane 5 development programme did not include adequate analysis and testing of the inertial reference system or of the complete flight control system, which could have detected the potential failure.**"

Software Failures or Disasters



- **Therac 25: Radiation Therapy Machine**
  - Used from 1985-87. Three died.
  - Note: Problems due to:
    - bad software design and development practices.
    - not explicitly to several coding errors that were found.
    - software was designed so that it was realistically impossible to test it in a clean automated way.
    - also, software flaw due to a *race condition* (which we'll study later for concurrent programming)

http://en.wikipedia.org/wiki/Therac-25

We have a "software crisis" (1968)

- "*Software Engineering*" term coined in 1968
  - NATO conference in Garmisch, Germany
- An "early" study of "problem" DoD projects
  - 47% of software delivered could not be used
    - Usually didn't meet **requirements**
  - 29% of funded software never delivered
    - Usually canceled due to **cost/schedule overruns**
  - 19% of software useful after extensive rework
    - Costs **36 times more** to fix problems after release

Definition of Software Engineering

- Fairley's definition:
  - Software engineering is the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates.

- Engineering versus science:
  - Production, practical, quality, maintenance, reuse, standards, teams, management, etc.

Software Engineering Is About…

- It's about:
  - Engineering
  - A systematic, careful, disciplined, scientific activity
  - Building software systems, particularly larger ones
    - Hacking or debugging until it works won't work
  - Modifying software systems over time
    - Not just focused on creating new applications.
- Programming may be fun, but Software Engineering should not be considered:
  - Easy, simple, boring or pointless

One Way to Think About It

- You want to cross a small creek – What do you do?

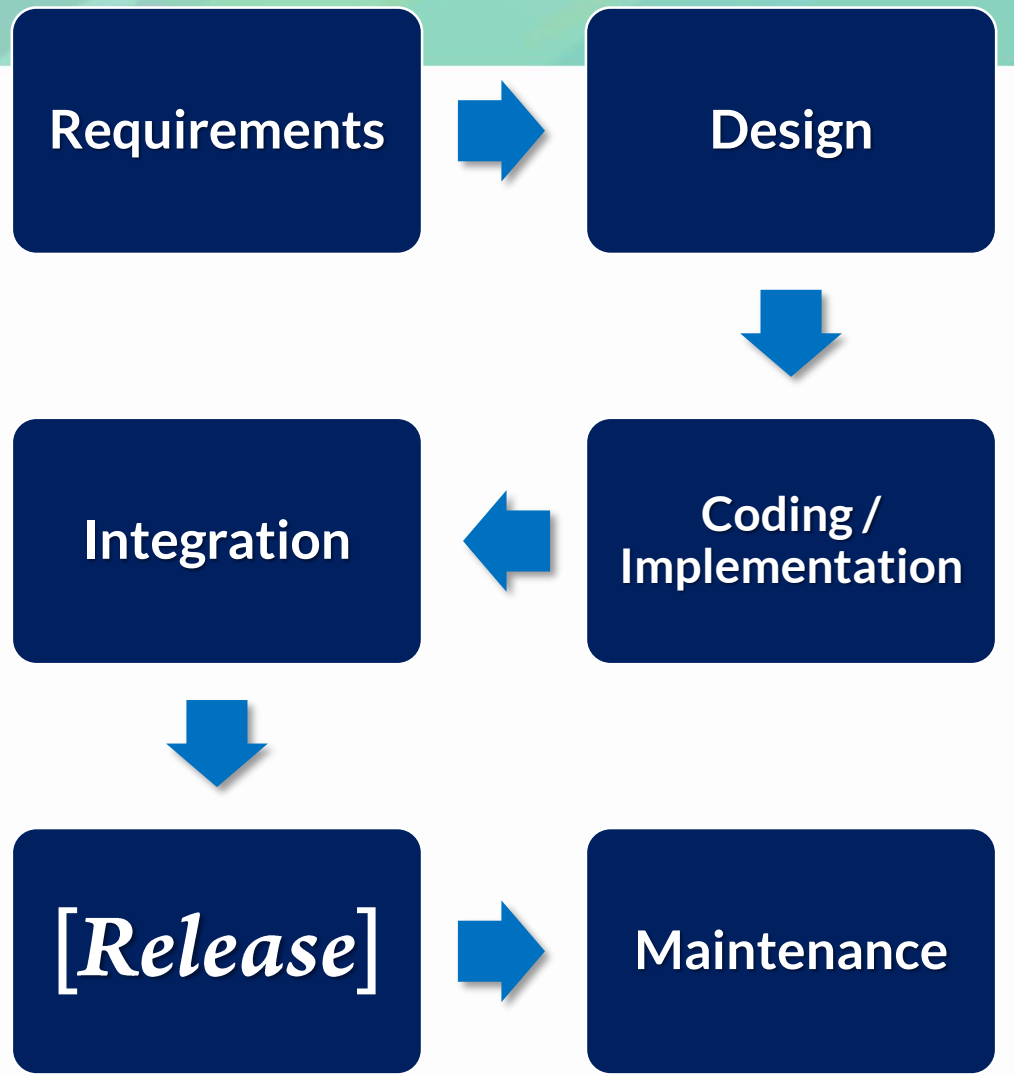
One Way to Think About It

- Build a bridge to cross a small creek
  - Cut down a tree. Drop a board across it.
- Build a bridge across the Golden Gate / Straits of Gibraltar
  - Need a **big** tree!! ☺
- What's different?
  - Size of project matters. Number of people involved. How long does it have to last. Risk/Consequences. Economics.
- "Programming in the large" vs. "Programming in the small"

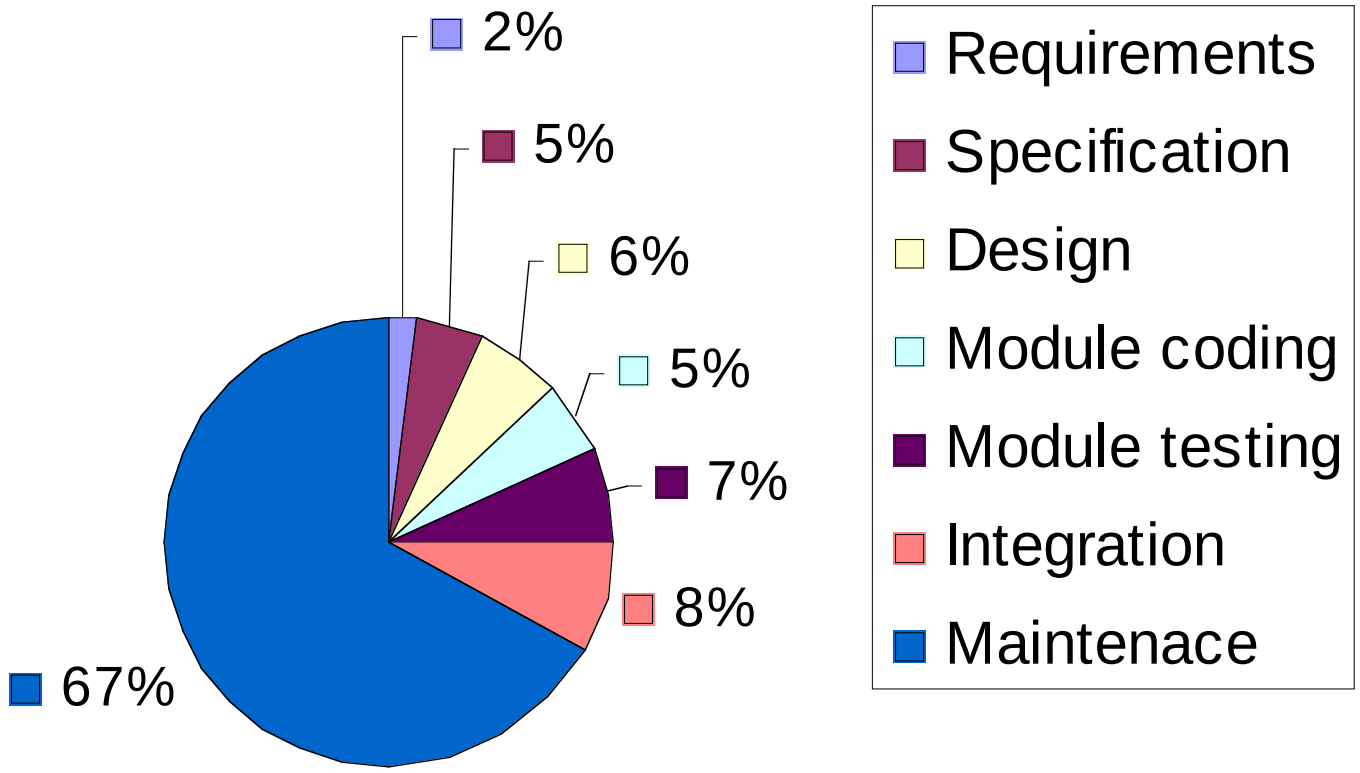
"Programming in the large" vs. "Programming in the small"

Software Lifecycle and Phases

- Stages or phases that are typical in software development, from "birth" to "death"
- **SW Development Lifecycle Phases:**
  1. Requirements ("Specification")
  2. Design
  3. Implementation ("Coding")
  4. Integration phase
     - Book calls this "testing phase".
       No! *Testing done throughout the lifecycle*
  5. Maintenance phase



Relative Cost per Phase

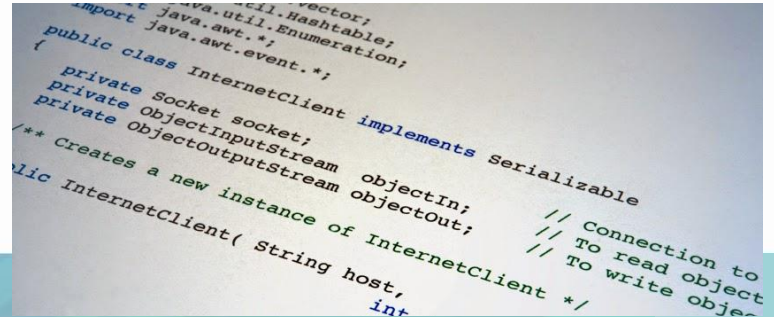


Analogy: Software Engineering is like Construction

Think about how buildings come to be:

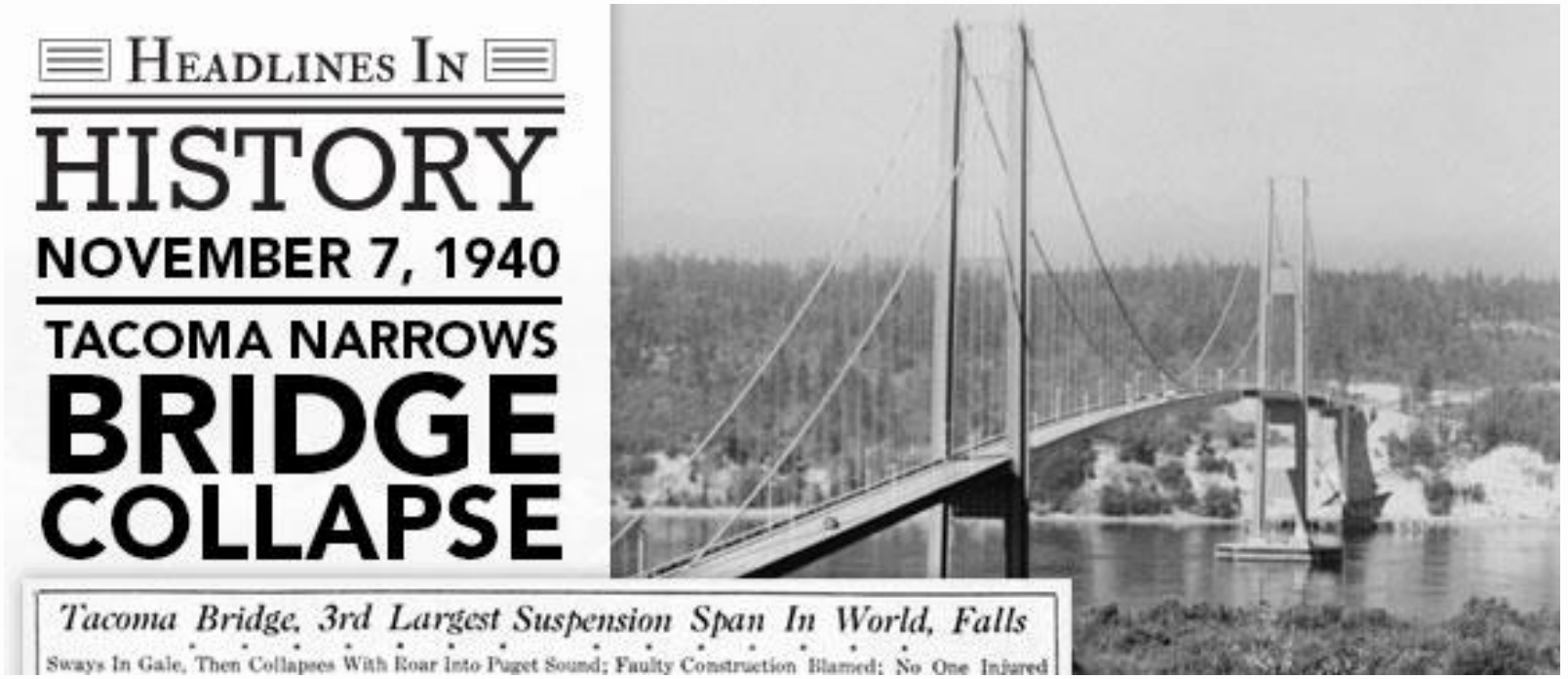
- Requirements
- Architecture
- Construction

- Differences?
  - Maintenance
    - Buildings don't change much
  - Design
    - Buildings really are less *complex*
      - Number of states
      - **Remove one brick (vs. remove one line of code!)**



… And sometimes Construction Is To Blame…

- https://www.youtube.com/embed/M5QNV3So7GM
- **Tacoma Narrows Bridge**: 3rd Largest Suspension Span In the World Falls



"Sways in Gale, Then Collapses With Roar Into Paget Sound; Faulty Construction Blamed; No One Injured"

Requirements

- Requirements are …
  - Statements of what the system should do (or what qualities it should have)
  - From the customer or client point of view
  - Not expressed in terms of a solution
- Requirements should be …
  - Should be clear, unambiguous statements of what must be done.
  - Must be consistent, complete, feasible, **testable**
  - **Not**: Description of how to do it (what classes, what methods, libraries, algorithms, etc).

Types of Requirements

- [1] **Functional requirements**
  - Describes a **function** or activity the system will carry out
  - Perhaps in response to input(s)
  - Describes the state of the system (or parts of the system) before and after an activity occurs
- [2] **Non-functional requirements**
  - What else might we want to describe?
  - **Qualities**: efficiency, reliability, usability, etc.
- [3] **Constraints**
  - On its **design**: Must run on certain platform, use particular components, etc.
  - On how its **built**: Must be built in a certain way (i.e. DOD standard process)
  - Directly from the client!
    - Explicit design-request from the client