15-712: Advanced Operating Systems & Distributed Systems

Introduction

Prof. Phillip Gibbons

Spring 2023, Lecture 1

Waitlist Status

- **As of Jan 17, 2023 at 10:30 pm: 39 registered, 4 on waitlist**
  - Possibly room for a few more students if many others drop
  - Please meet me after class

- **Admittance priority:**
  - CSD PhD, ECE PhD, other SCS PhD
  - CS Masters, CS Undergrads
  - ECE Masters, ECE Undergrads
  - other Masters, other Undergrads

- **Priority among Masters students (and Undergrads) based on relevant courses taken (e.g., 213/513/613, 15-410/610) and grades obtained**

Today's Topics

- **Course Overview**
  - No slides, just a walk through of the key points on the course webpages

- **Discussion of 2 Wisdom Papers**

The Mythical Man-Month

Fred Brooks 1975

1931-2022
Turing Award winner

- **Why programming projects are hard to manage**

"Good cooking takes time. If you are made to wait, it is to serve you better, and to please you."– Antoine's chef

- **Tar Pit:**
  - Program -> Programming Product (tested, documented) = 3x
  - Program -> Programming System (APIs, meet resource budget, inter-component testing) = 3x
  - Total = 9x programming time

- **Woes of Programming: must perform perfectly, authority below responsibility, dependent on others code, debugging is tedious/ slow to converge, program feels obsolete by time it is done**

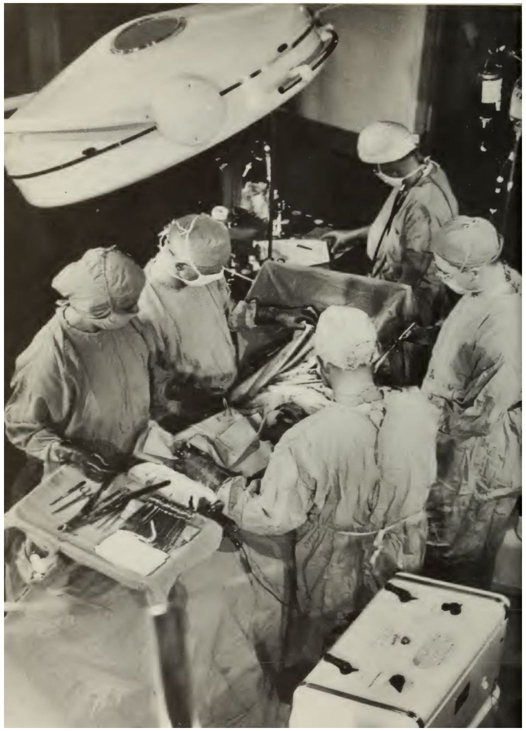
Mythical Man-Month

- **Optimism: Techniques of estimating time are poorly developed**
- **Fallaciously confuse effort (months) with progress**
  – must consider project communication overheads
- **SW managers lack the courteous stubbornness of Antoine’s chef**
  – false scheduling to match a patron’s deadline
- **Schedule progress is poorly monitored**

Brook’s Law: “Adding manpower to a late software project makes it later”

The Surgical Team

- **Among experienced programmers, best are 10x productive and code is 5x faster/smaller**
  - But small teams will take too long



- **[Harlin Mills] Team of 10:
  Surgeon, copilot, administrator, editor,
  2 secretaries, program clerk, toolsmith, tester,
  language lawyer (knows performance hacks)**

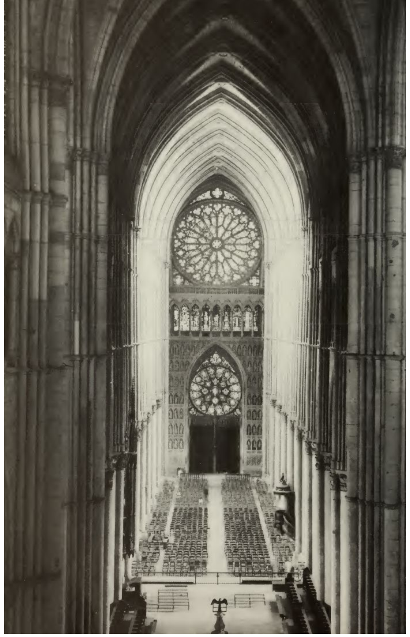
- **Hard to scale up to larger teams**

Aristocracy vs. Democracy

- Conceptual integrity is THE most important consideration in system design

- Ratio of function to conceptual complexity is the ultimate test of system design

- Division of labor between architecture (complete/detailed specification of the user interface) and implementation
  - what vs. how
  - can proceed somewhat in parallel



Second-System Effect

- An architect's first work is apt to be spare and clean
- But second systems tend to go overboard

Passing the Word

- **Specifications should be both formal definitions & prose definitions**
  - don't use an implementation as specification
- **Weekly half-day conferences**
- **(Semi-)annual two-week courts among larger group**
  - Before each manual freeze
- **2 implementations!**
  - Enforces fidelity to the specification, since fixing incorrect implementation is better than unfixing correct implementation

Productivity & Size

- Interruptions while coding are bad

- Operating systems 3x slower to code than compilers, Compilers 3x slower than batch application programs

- Write two versions of each important routine: the quick and the “squeezed”

- Representation (data structure) is the essence of programming

Plan to Throw One Away

- …you will anyway

- Plan the system for change
  - modular design, versions

- Have a Technical Cavalry at your disposal

Tacoma Narrows Bridge, 1940

- Program Maintenance: Cost of maintaining a widely-used program is typically 40% or more of the cost of developing it

"Program maintenance is an entropy-increasing process, and even its most skillful execution only delays the subsidence of the system into unfixable obsolescence"

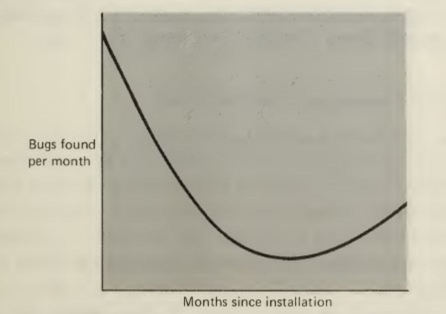


Fig. 11.2 Bug occurrence as a function of release age

The Whole and the Parts

- The most pernicious and subtle bugs are system bugs arising from mismatched assumptions made by authors of various components

- Use top-down design with stepwise refinement

- Many poor systems come from an attempt to salvage a bad basic design and patch it with all kinds of cosmetic relief

- Half as much code in scaffolding (for debugging) as in product

Hatching a Catastrophe

- How does a project get to be a year late?

  …One day at a time

- During the activity, (rare) overestimates of duration come steadily down as the activity proceeds

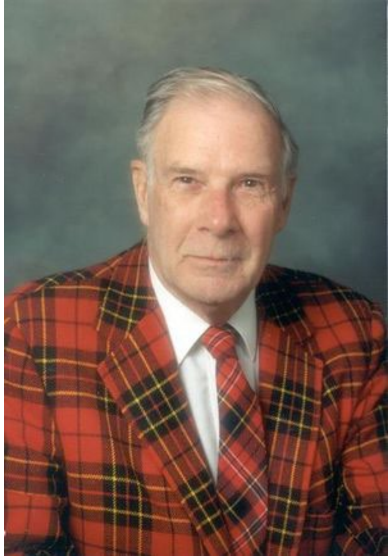
- Underestimates do not change significantly during the activity until about 3 weeks before the scheduled completion

- Do critical path planning analysis (PERT chart)

- Self-document programs: comment the source code (!)

You and Your Research

Richard Hamming 1986



1915-1998

- **Hamming distance**
- **Hamming codes (first error correcting codes)**
- **Turing Award winner 1968**
- **"The purpose of computing is insight not numbers"**

Q: Why do so few scientists make significant contributions and so many are forgotten in the long run?

How to be a Great Scientist

- "Luck favors the prepared mind" – Pasteur
- As teenagers, they had independent thoughts & the courage to pursue them
- Key Characteristic: Courage
- Do best work when they are young professionals
  - After do good work, put on all sorts of committees
  - When you are famous it is hard to work on small problems (Fail to plant the acorns from which the mighty oaks grow)
  - The IAS at Princeton has ruined more good scientists than any institution has created

How to be a Great Scientist

- **People are often the most productive when working conditions are bad**

- **Most great scientists have tremendous drive**
  - must be intelligently applied

- **Knowledge and productivity are like compound interest**

- **Great scientists tolerate ambiguity well**

- **…are completely committed to their problem**
  - keep your subconscious starved so it has to work on your problem

How to be a Great Scientist

- **What are the important problems in your field?**
  - and must have plan of attack

- **Set aside a "Great Thoughts" time**

- **When an opportunity opens up, get after it and pursue it**

- **He who works with the door open gets all kinds of interruptions, but he occasionally gets clues as to what the world is and what might be important**

- **Never again solve an isolated problem except as characteristic of a class**

- **Do your job in such a fashion that others can build on it**

How to be a Great Scientist

- **Need to sell your work, via good writing, formal talks, and informal talks**
  - Make talks be more big picture

- **Is the effort to be a great scientist worth it?**

- **Personality defects such as wanting total control, refusing to conform to dress norms, fighting the system rather than take advantage of it, ego, anger, negativity**
  - Let someone else change the system

- **Know yourself, your strengths and weaknesses, & your bad faults**

How to be a Great Scientist

- Should get into a new field every 7 years
- The bigger the institutional scope of your vision, the higher in management you need to be
- In the long-haul, books that leave out what's not essential will be most valued
- Do library work to find what the problems are
- Refuse to look at any answers until you've thought the problem through carefully how you would do it, how you could slightly change the problem to be the correct one
- Choose the right people to bounce ideas off of

To Read for Friday

"Hints for Computer System Design" (write summary)
Butler Lampson 1983

"End-to-End Arguments in System Design" (no summary)
Jerome Saltzer, David Reed, David Clark 1984

Optional Further Reading:

"The UNIX Time-Sharing System"
Dennis Ritchie & Ken Thompson 1974