The Dawn of the Rise of the Planet of Software Engineering

“THE TAO GAVE BIRTH TO MACHINE LANGUAGE.

MACHINE LANGUAGE GAVE BIRTH TO THE ASSEMBLER.

THE ASSEMBLER GAVE BIRTH TO THE COMPILER.

NOW THERE ARE TEN THOUSAND LANGUAGES.

EACH LANGUAGE HAS ITS PURPOSE, HOWEVER HUMBLE.
EACH LANGUAGE EXPRESSES THE YIN AND YANG OF
SOFTWARE. EACH LANGUAGE HAS ITS PLACE WITHIN
THE TAO.

BUT DO NOT PROGRAM IN COBOL IF YOU CAN AVOID IT.”

-- GEOFFREY JAMES
Many Eons Ago...

- Early in the history of software development, there was no “software engineering” as such

- Couple of reasons for this:
  - Software was built/run using **batch processing**
  - Hardware was constantly changing → early on coding done in **assembly** specific to hardware
Batch Processing

- Software was built/run using **batch processing**
  - Make program on punch cards
  - Put into machine to compile/run
  - Either get the program’s results...or find out that you have errors...
- This made software development very, VERY slow...

[Image: http://www-03.ibm.com/ibm/history/ibm100/images/cep/Q96973285093Q79/us__en_us__ibm100__punched_card__hand_cards__620x350.jpg]
• Mid-50’s – mid-60’s → **higher-level languages** became the norm
  - Could port to different architectures
  - *Examples*: FORTRAN, COBOL, ALGOL

• Mid to late 60’s → **interactive, time-sharing machines**
  - Could compile/run code at terminal

• Suddenly, there’s a LOT more software...
The “Software Crisis” was identified in the mid/late 60’s

- Projects over time and budget (or never completed at all)
  - Poor estimation of how long things would take
  - Poor monitoring of progress
- Projects poorly managed (or not at all)
  - Assumption that people and months are interchangeable
  - Adding manpower when behind schedule
- Software did not meet requirements and/or was of poor quality
  - ...sometimes with fatal results...
Example: Therac-25

- Therac-25 = radiation therapy machine
  - Software based on previous versions
  - *Never bothered to check whether current software was working properly*
  - At least 6 people given massive overdoses of radiation (1980’s)
Edsger Dijkstra on the “Software Crisis”

- In his Turing Award Lecture “The Humble Programmer”, Dijkstra talked about why we had/have a “software crisis”:

  - “the major cause is... that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming had become an equally gigantic problem.” (emphasis mine)
Software Engineering is Born

- 1968 – first NATO Software Engineering Conference (Germany)
  - Term “software engineering” coined
    - ...although some dispute that claim: https://bertrandmeyer.com/2013/04/04/the-origin-of-software-engineering/
  - Trying to address “software crisis”
  - Recognized need to some formalized **process** of software development
    - Sometimes just called “software process”

Fred Brooks and the Mythical Man-Month

“THE BEARING OF A CHILD TAKES NINE MONTHS, NO MATTER HOW MANY WOMEN ARE ASSIGNED.”

-- FRED BROOKS
Fred Brooks

- Dr. Frederick P. Brooks, Jr.
  - Project manager of IBM System/360 (hardware)
    - Coined term “computer architecture”
  - Manager of OS/360 software project
    - Over 1000 people working on it
    - The problems and issues encountered resulted in his book “The Mythical Man-Month” (1975)
  - Won Turing Award in 1999
  - Recently retired as professor of CS at UNC Chapel Hill
The Problems with Software Development

- In his book, he cites a few reasons why software projects fail:
  - Poor estimation
  - Assuming people and months are interchangeable
  - Adding manpower to a late project
  - “Gutless estimating”
  - Poorly monitored progress
Poor Estimation

- Programmers are optimists
  - **False Assumptions:**
    - “All will go well”
    - “It will take as long as it SHOULD take”

- *At the time:* not much published data on successful projects and methods

- *Present day:* we have better models for estimating project length (e.g., COCOMO II), but even then nothing is bulletproof
Because we assume nothing will go wrong → we don’t allocate enough time for testing and debugging
The Man-Month

- **Man-month** = 1 person working for 1 month

- **Idea**: people and months were interchangeable
  - “Surely, if 1 programmer can finish it in 6 months, then 6 programmers can finish it in one month!”
The “Mythical Man-Month”

- Fred Brooks in his book “The Mythical Man-Month” points on that people and months are not-interchangeable if the task cannot be perfectly partitioned
  - Similar to problem with multi-core processing

- Adding more people means → more lines of communication

- Adding more people LATER in the project:
  - More lines of communication
  - More training time
  - Possible poor partitioning of remaining work

- **Brook’s Law**: Adding manpower to a late software project makes it later.

\[
C(n,2) = \frac{n!}{2!(n-2)!}
\]
Gutless Estimating

- No matter how good are estimation approaches are, sometimes we will promise more than we can deliver
  - *Example used*: an omelette that SHOULD have been done in two minutes...

- At least two options:
  - “Turn up the heat”
    - Rush the project → end up with lower-quality product
  - Be honest
    - Tell the customer the bad news

- There is sometimes a third option:
  - Discuss with customer alternative product with fewer feature that CAN be completed in original time-frame

References
References

- [http://www.vikingcodeschool.com/software-engineering-basics/a-brief-history-of-software-engineering](http://www.vikingcodeschool.com/software-engineering-basics/a-brief-history-of-software-engineering)
- [https://www.dagstuhl.de/Reports/96/9635.pdf](https://www.dagstuhl.de/Reports/96/9635.pdf)