A HISTORY OF SOFTWARE ENGINEERING RESEARCH

for CSCI-5828, Spring Semester

Prepared by Bradley Cooper

Executive Summary
Software Engineering Research

Software consists of “the programs and other operating information use by a computer.” And the term **Engineering** is defined by Merrium-Webster as “the application of science and mathematics … [towards] the design and manufacture of complex products.”

The IEEE defines **Software Engineering** as:

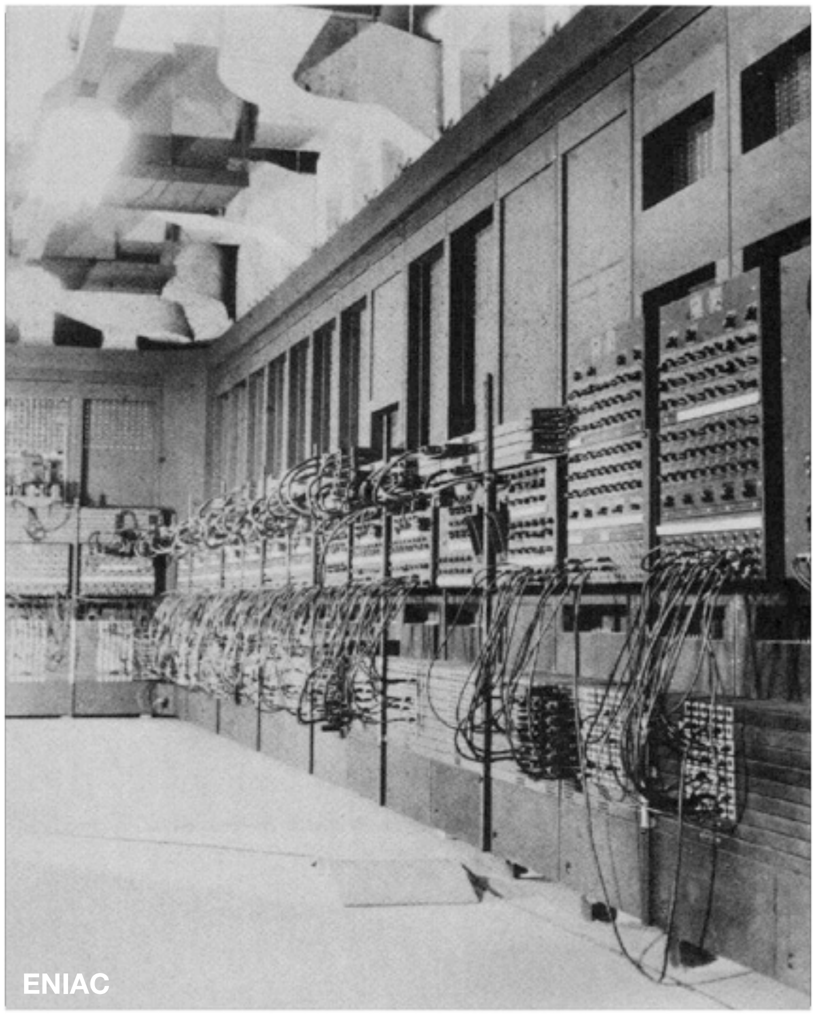
> *“The Application of systematic, Disciplined, Quantifiable Approach to the Development, Operation and Maintenance of Software; that is, the application of engineering to software.”*
>
> *- IEEE, Definition of Software Engineering*

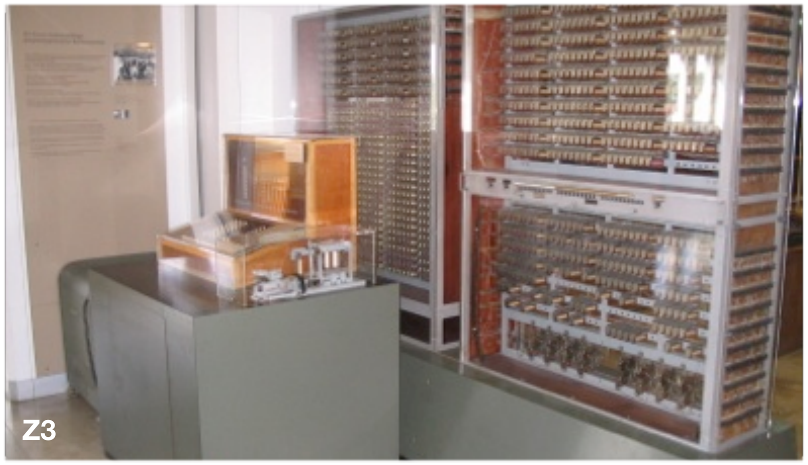
The term Software Engineering was first coined at the ‘NATO Software Engineering Conference’ in 1968. But, in reality, people have been ‘engineering’ software further back than that.

This presentation hopes to outline a brief history of the development of Software Engineering from the early days of the 20th century, to today. Taking a quick look at the hardware, concepts, methods and drives along the way.

Presentation Goals

I. Intro Software Engineering
II. Early Hardware
III. Programming Languages
  i. The Why
  ii. Early Languages
  iii. Object Orientation
IV. Operating Systems
  i. Overview
  ii. OS/360
  iii. UNIX and Beyond
V. Software Crisis
  i. NATO Conference
  ii. Silver Bullet
VI. Development Life Cycles
  i. Traditional
  ii. Agile
VII. Changing Environments
  i. Micro Computer
  ii. Distributed Systems
  iii. The Internet
VIII. Recap
IX. Further Reading

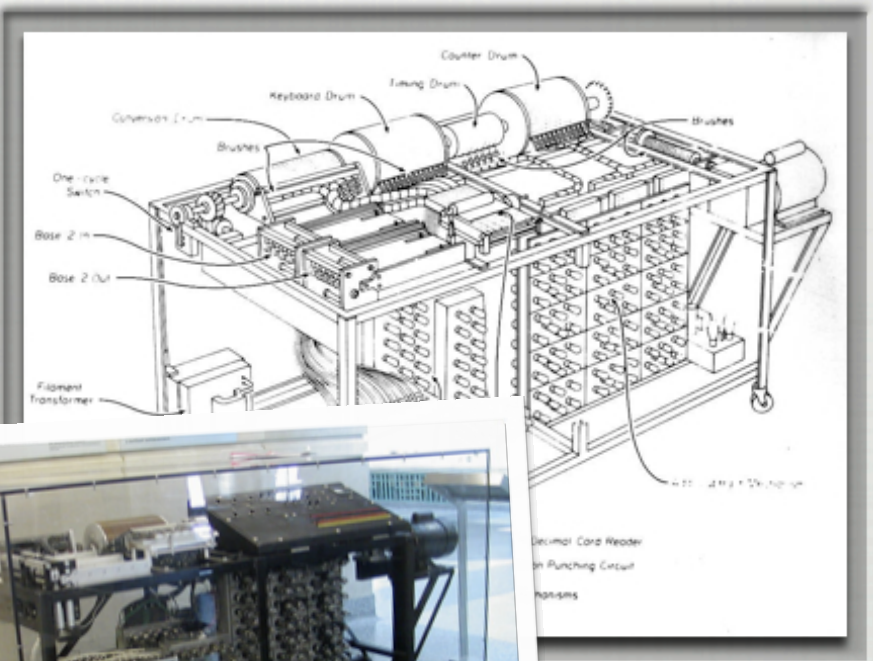




HARDWARE

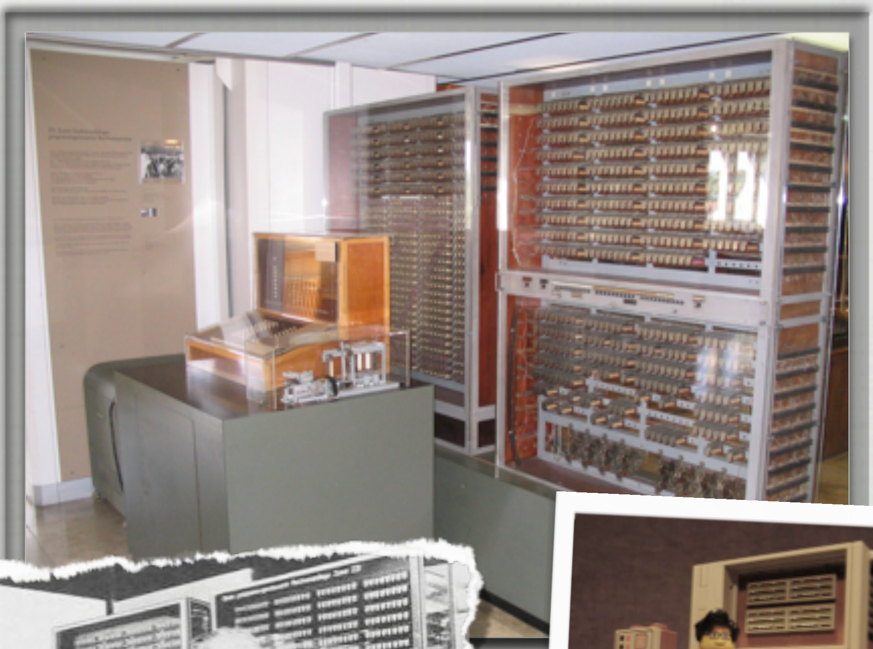
ATANASOFF-BERRY COMPUTER

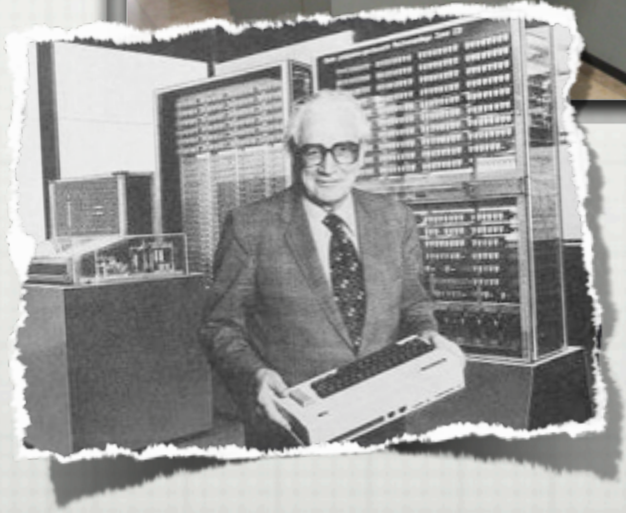
- **First Operational** in 1937 (tested 1942)
- **Claim to Fame:**
  - First ‘Electronic Digital Computing Device’
- **Designers:**
  - John Atanasoff and Clifford Berry
- **Affiliation:**
  - Iowa State College
    (now Iowa State University)
- **Intent:**
  - Linear equation solver
- **Basic Specs:**
  - Binary Arithmetic
  - Vacuum Tubes
  - Regenerative Capacitive Memory
  - Not programmable



Z3

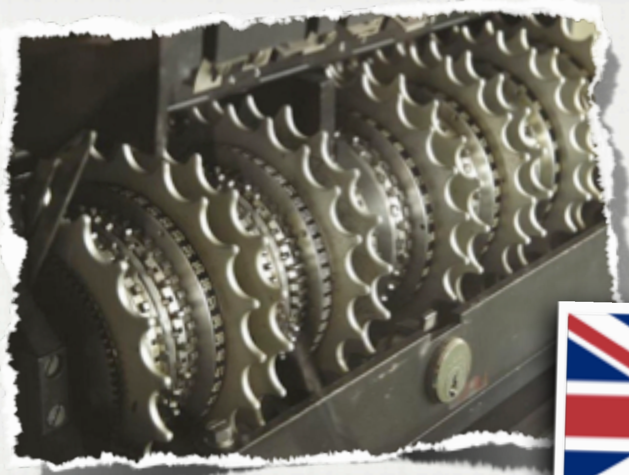
- **First Operational** in 1941
- **Claim to Fame:**
  - First Programable ‘Complete’ Computer
- **Designers:**
  - Konrad Zuse
- **Affiliation:**
  - ZUSE Apparatebau (German company)
  - German Government
- **Intent:**
  - Wing Flutter Calculations
- **Basic Specs:**
  - Binary Arithmetic
  - Electromechanical Relays
  - External Program Storage (tape)
  - Programmable



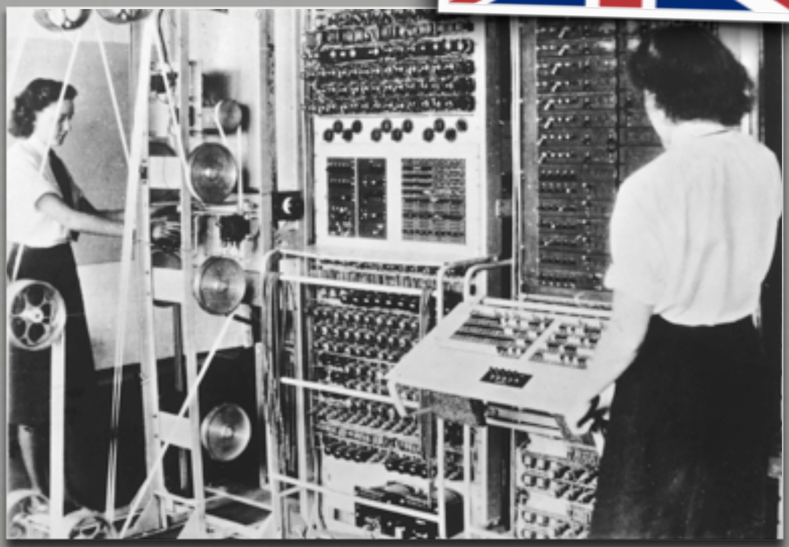


COLOSSUS

- **First Operational** in 1943
- **Claim to Fame:**
  - First Electronic, Digital and Programmable Computer
- **Designer:**
  - Tommy Flowers (& others)
- **Affiliation:**
  - Post Office Research Station
  - British Government
- **Intent:**
  - Cryptanalysis of German Communications *(During World War II)*
- **Basic Specs:**
  - Binary Arithmetic
  - Thermionic Valves (vacuum tubes)
  - Programmable

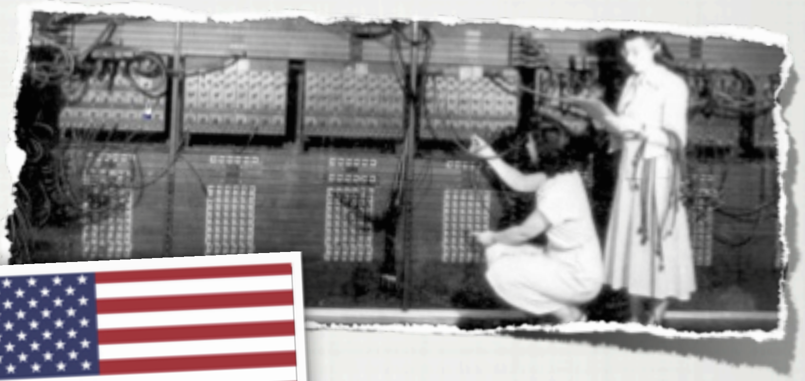


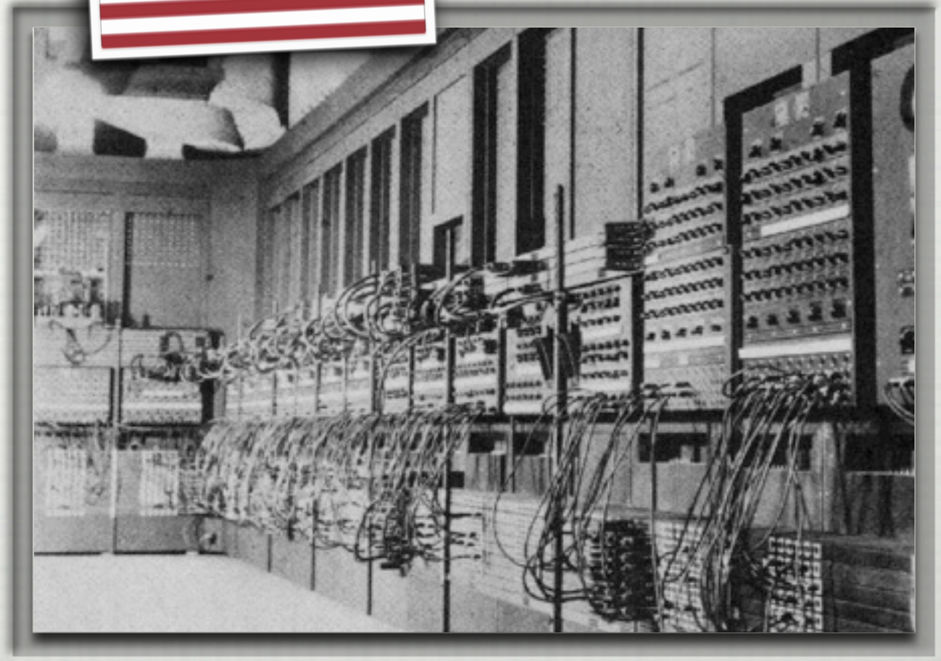
<- German Lorenz Encryption Machine



ENIAC

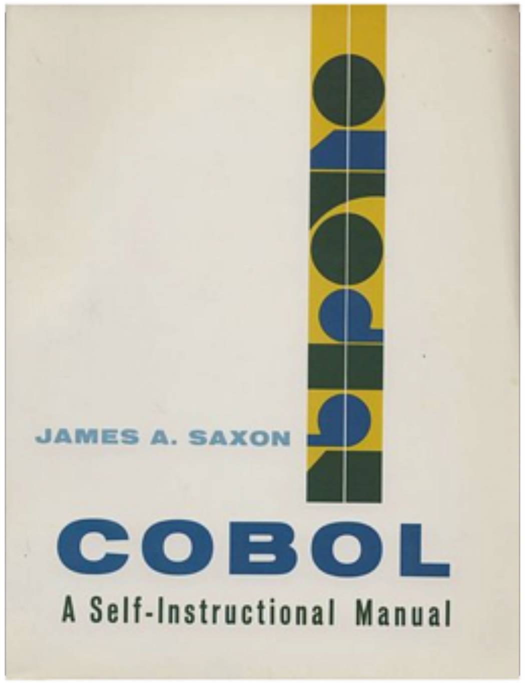
- **First Operational** in 1946
- **Claim to Fame:**
  - First ‘Designed’ for Turing-Complete Electronic Digital Computer
- **Designer:**
  - John Mauchly, J. Presper Eckert
- **Affiliation:**
  - United States Government
- **Intent:**
  - Artillery Firing Tables,
- **Basic Specs:**
  - Decimal Arithmetic
  - Thermionic Valves (vacuum tubes)
  - Programmable





Java™

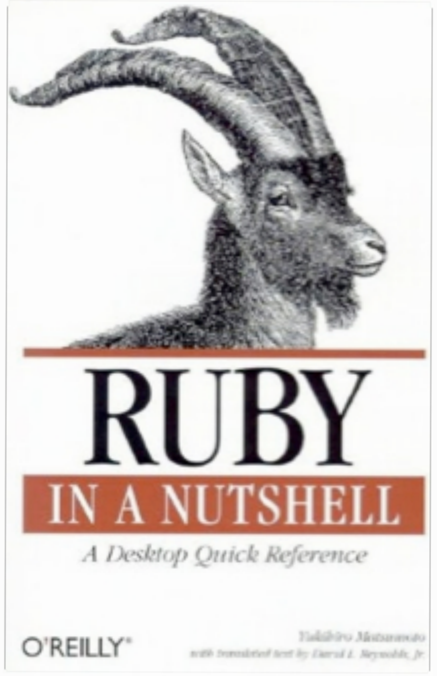
Fortran



JAMES A. SAXON

COBOL

A Self-Instructional Manual



RUBY

IN A NUTSHELL

A Desktop Quick Reference

O'REILLY®

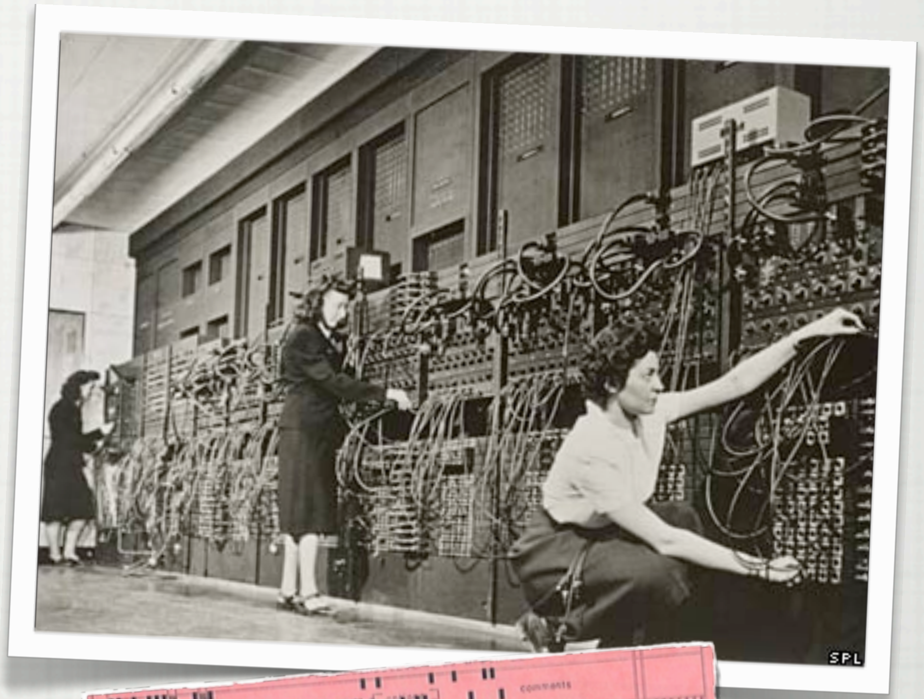
Yukihiro Matsumoto

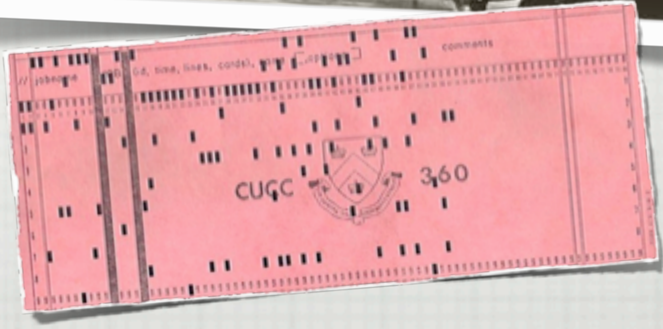
with translated text by David L. Reynolds, Jr.

PROGRAMMING LANGUAGES

A NEED FOR PROGRAMMING LANGUAGES

- Initially, if you or the institution you were part of desired to solve a problem with computers, you would design a machine that would be specifically tailored to solve that task.
- Even early on, the complexity of systems like ENIAC would require teams of technicians weeks to reprogram.
- Programming languages offered a level of abstraction that let the programmer step back, making the act of programming more straightforward and accessible.

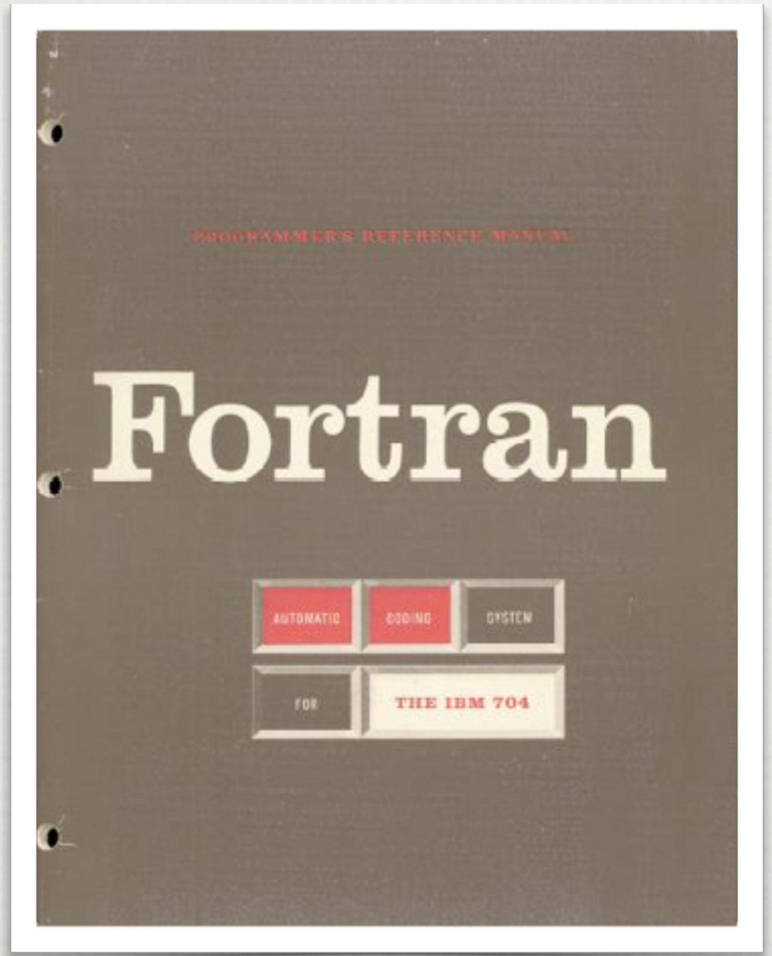




LEVELS OF ABSTRACTION

- **Low-Level Languages**
  (low abstraction)
  - **Machine Code**
    - Code that is directly interpreted by the processor of a machine
    - Deals most directly with hardware
  - **Assembly**
    - Typically regarded as low-level
    - Still hardware specific
- **High-Level Languages**
  (high abstraction)
    - High level languages make it possible to use natural language elements in programming.
    - Also, they are more portable.
    - For example, UNIX written in assembly in 1970 and was rewritten in C in 1972, making it as portable as the C programing language.

EARLY LANGUAGES

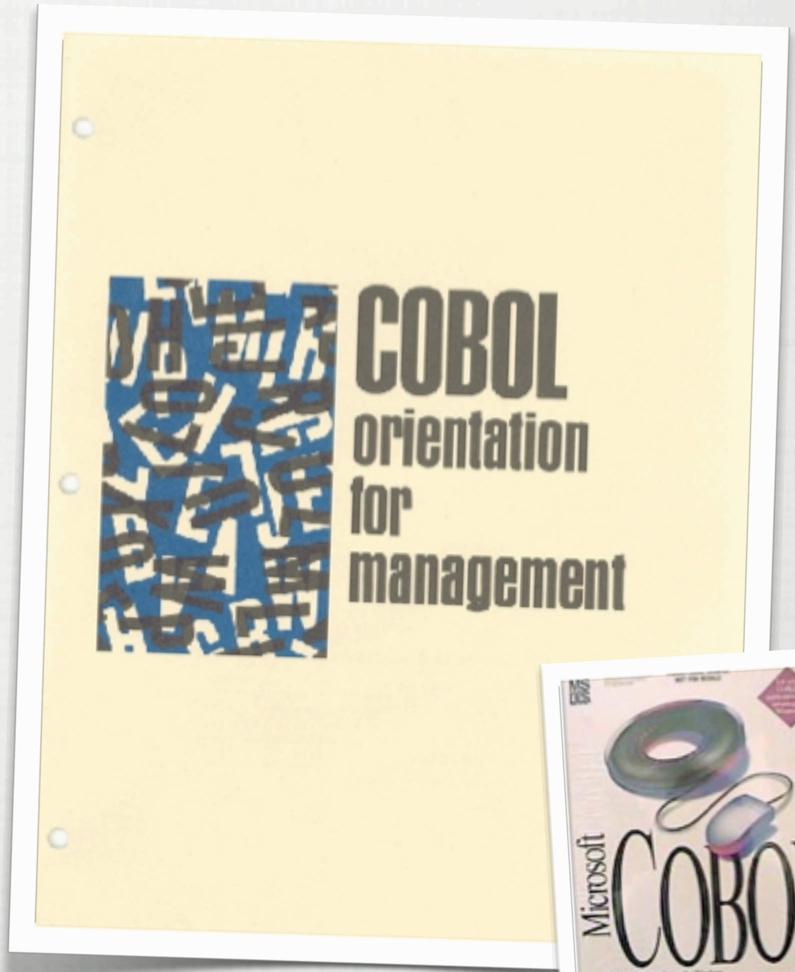


- High-level programming languages began to show up on the scene in the 1950s

- **FORTAN**
  - Began at IBM in 1953, FORTRAN delivered the first compiler (and all associated materials) by 1957.
  - Used for Numerical Computation and Scientific Computing
  - Programmers were initially skeptical of this high-level language, until the benefits were made clear in the form of efficiency gains

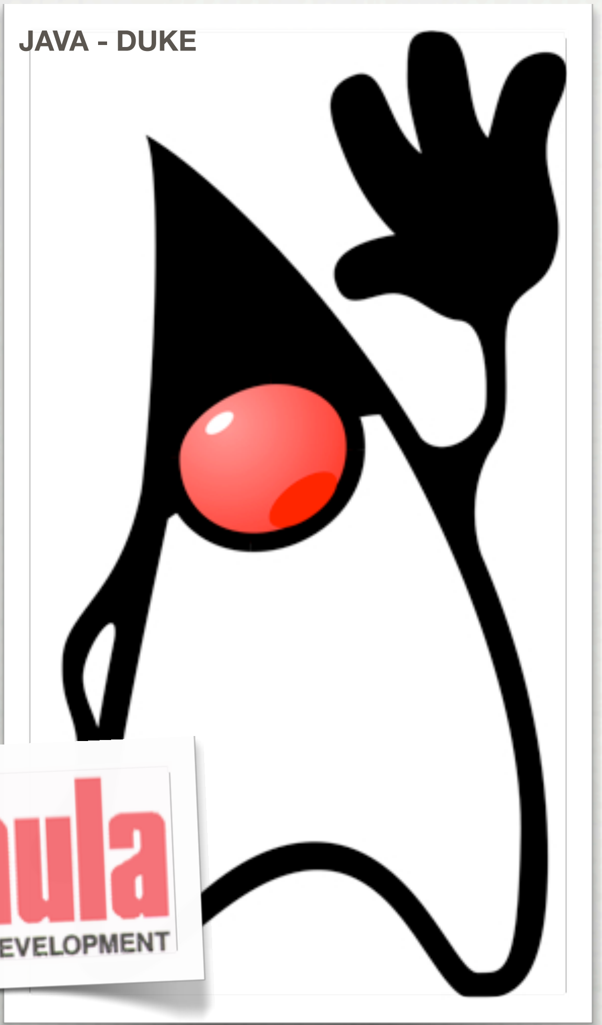
EARLY LANGUAGES

- **Lisp**
  - Second of the early high-level languages, Lisp was developed to be a practical mathematical language
  - Popular dialects are Scheme, Common Lisp and Clojure
- **ALGOL**
  - Built in 1958, ALGOL was in some ways a response to FORTRAN
  - It would later influence many languages, including Pascal and C
- **COBOL**
  - Designed for Business in 1959
  - COBOL 2002 was the most recent release



Last MS-DOS version of Microsoft COBOL

OBJECT ORIENTED PROGRAMING



- Object oriented aspects have been built into various languages from the late 50s on
- However, the first primary featured object oriented language is considered to be **Simula**, developed in 1967
- Object-oriented properties help to further the abstract qualities programming languages.
- In general, object oriented practices contribute to software engineering the ability to encapsulate and modularize complex functionality.
- Inheriting from other objects and a focus on code reuse are cornerstones

MICRO
SOFT

Windows

UNIX

Mac OS

IBM®

OS/360

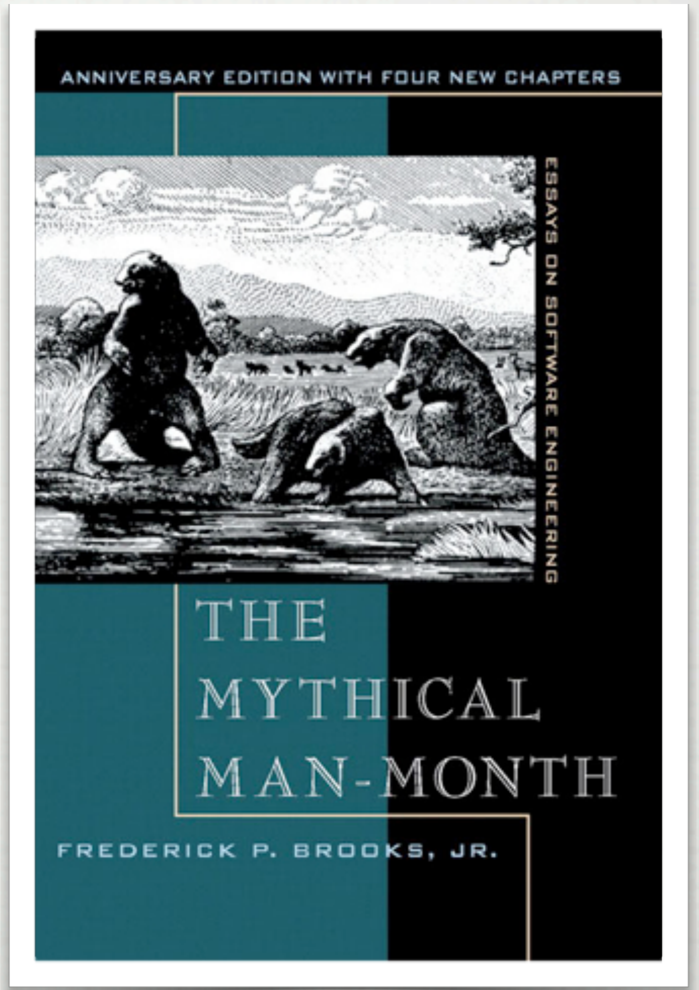
OPERATING SYSTEMS

OPERATING SYSTEM OVERVIEW

- With the development of more and more complex systems of software, users needed a better way to interact with their systems
- GM was arguably the first company to construct their own operating system in 1956, calling it **GM-NAA I/O**, for their IBM 704.
- Most mainframe users tended to construct their own OS, typically for each system in use.
- IBM's **OS/360**, sold from 1964 to 1978, was one of the first operating systems that separated architecture and implementation.

OS/360

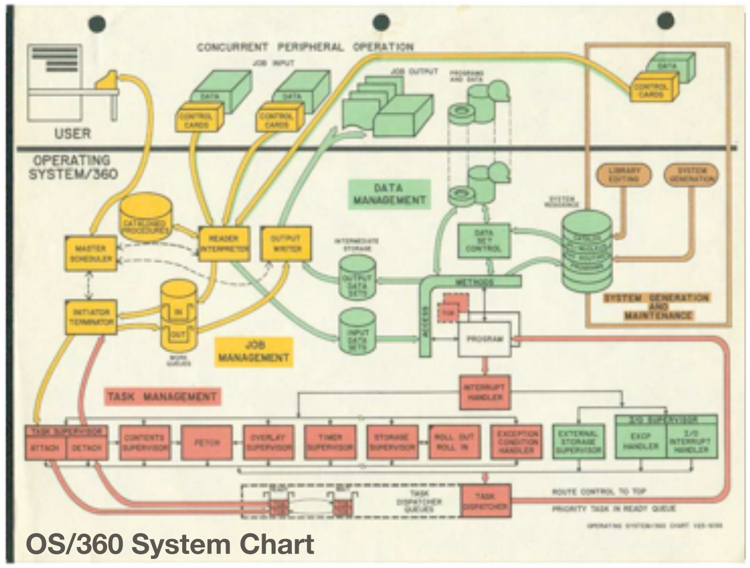
- The **OS/360** line of computers and operating systems made it possible, to write applications capable of being run on multiple systems
- OS/360 also represented one of the largest and most ambitious software engineering projects of all time
- At it's peek, more than 1000 employees worked on the project, and IBM spent nearly half a billion dollars in development
- Fred Brooks would later publish his famous book "The Mythical Man-Month" on his experience managing the project


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

UNIX AND BEYOND

- In 1970, Unix was written by engineers at Bell Labs
- However, the 1958 antitrust settlement forbad AT&T from entering the computer industry
- Unix began being licensed copiously.
- Unix has found it’s way into many applications from Government, Educational models and modern operating systems (Mac OS X, iOS, Linux)
- Unix, Windows and other Unix-based operating systems have come to form the basis of our modern software infrastructure.

NATO



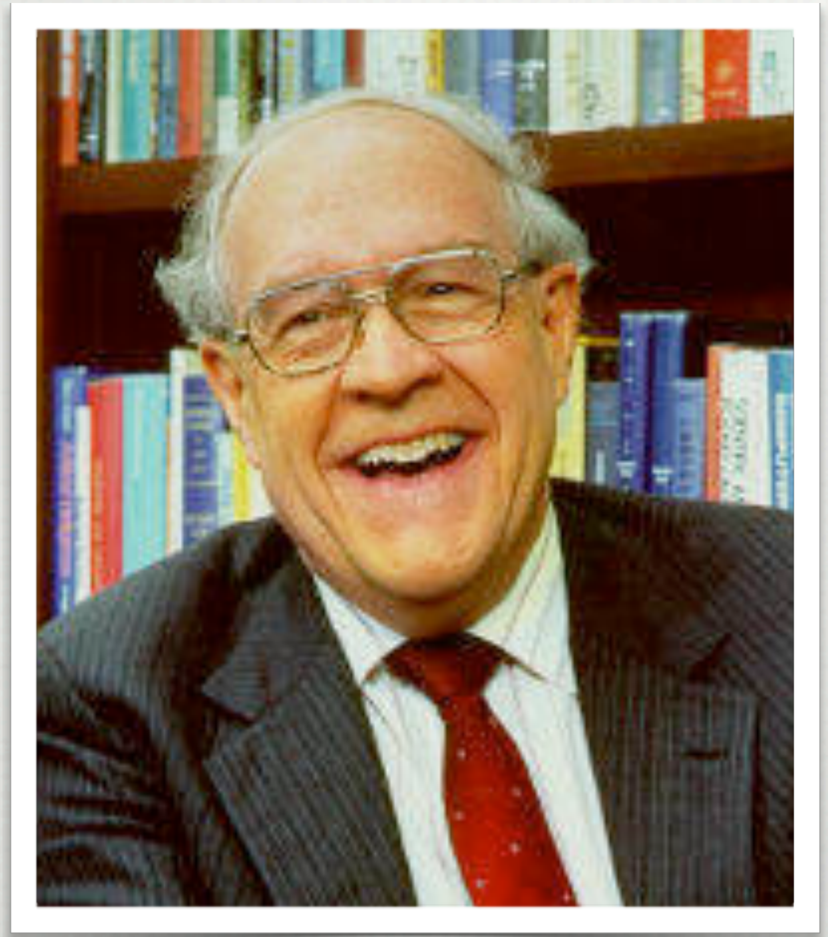
OS/360 System Chart

SOFTWARE CRISIS

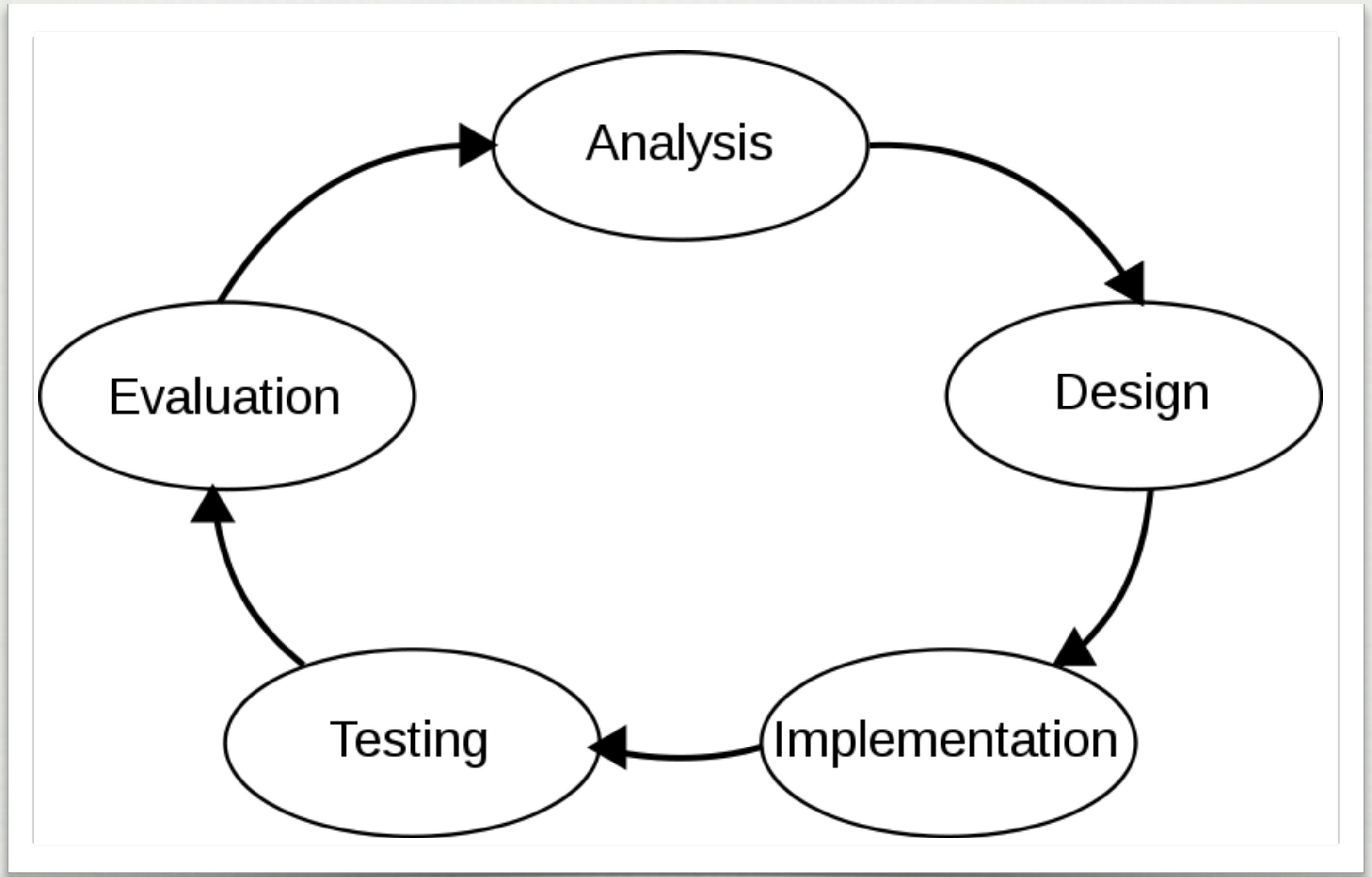
Software Crisis

- As computer hardware functionality and complexity continued to progress in leaps and bounds, it became clear that the number one problem associated with so much powerful hardware is the software.
- NATO convened the 1968 Software Engineering Conference.
- Delegates discussed several resultants due to hardware complexity and software development difficulties:
  - Project over-runs
  - Budget over-runs
  - Low quality, and many more
- At the time of the crisis, the OS/360 system had just been completed, one year and many millions of dollars over budget
- It was clear then, and in the next several years, that methods were needed

SILVER BULLET



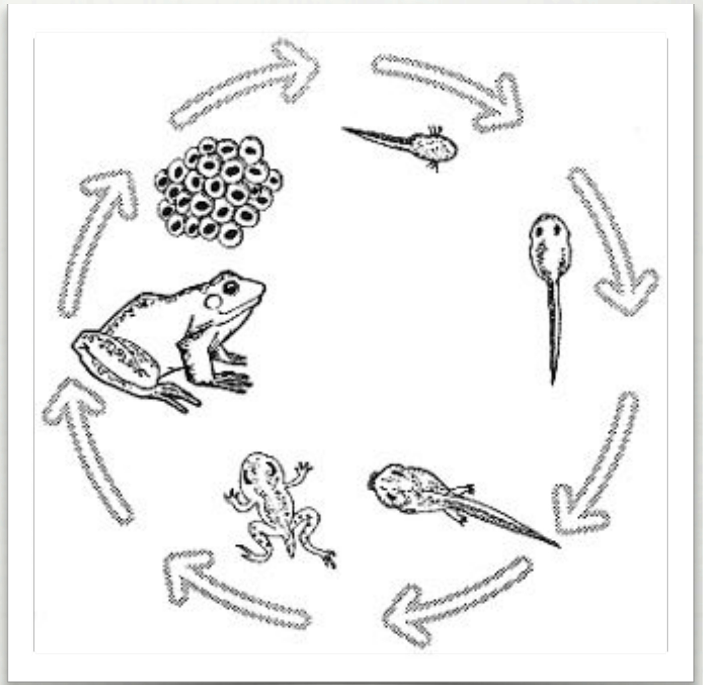
- For many years after the NATO conference, much of Software Engineering and Computer Science became about finding that one technique to fix all the problems
- Fred Brooks called the ‘fixer’ a Silver Bullet in his 1986 paper “No Silver Bullet”
- Brooks claims that there is no easy fix that will solve all the computer engineering problems
- His contention, is that as a whole, our myriad techniques will add up to advances of large factors, but most likely no ONE thing will do it quickly



DEVELOPMENT LIFE CYCLES

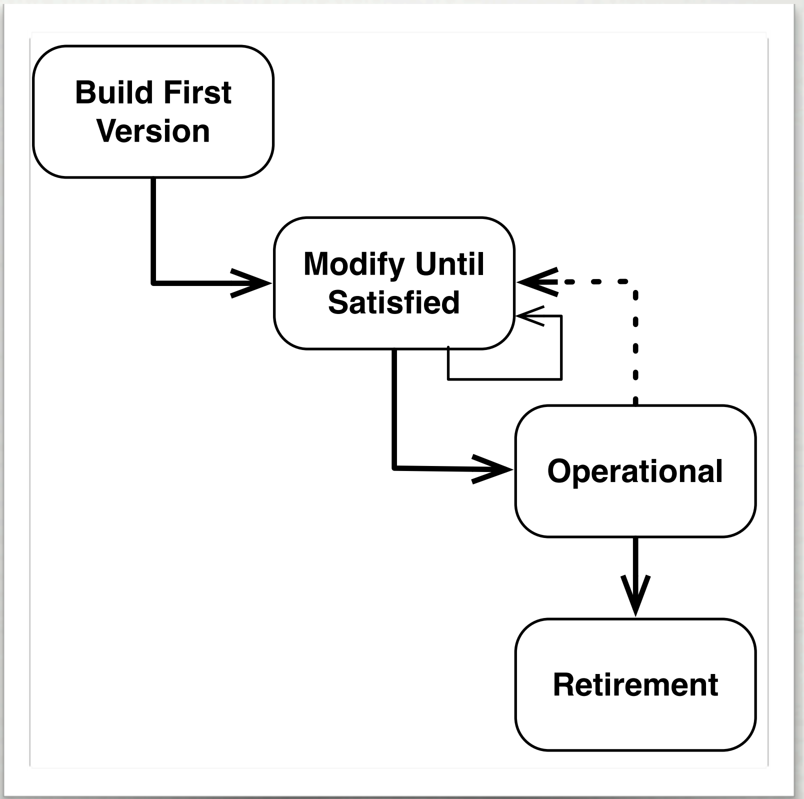
TRADITIONAL LIFE CYCLES

- Software Life Cycles offer several benefits:
  - Metrics with which to judge system progress
  - Tools to help estimate time per task
  - Aspects of the process that can be reused or avoided in the future
- Most life cycles use forms of the same 5 steps (previous slide), but differ on emphasis or repetition of those steps
- Here are a few (of many) life cycles
  - Code and Fix
  - Waterfall
  - eXtreme Programming (XP)

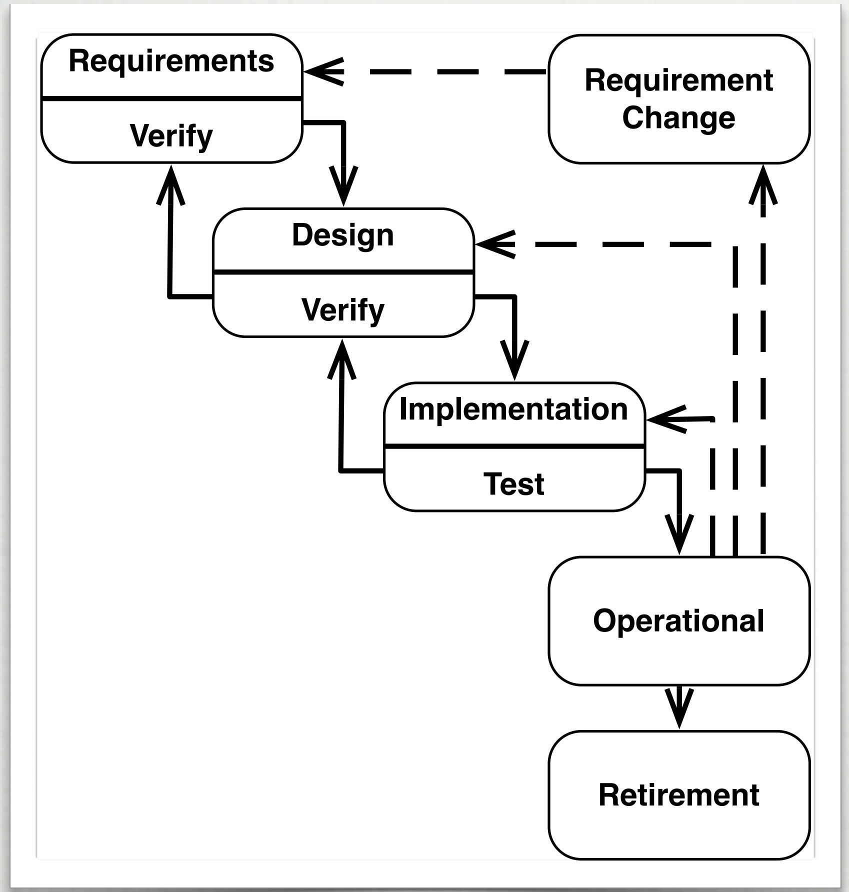

A LITTLE LIKE THIS, BUT WITH SOFTWARE

CODE AND FIX

- Code and fix is by far the easiest method of producing small programs
- Major problems arise however if any reasonable amount of complexity is involved
- It’s also fairly difficult to distribute a project amongst a group with this model.



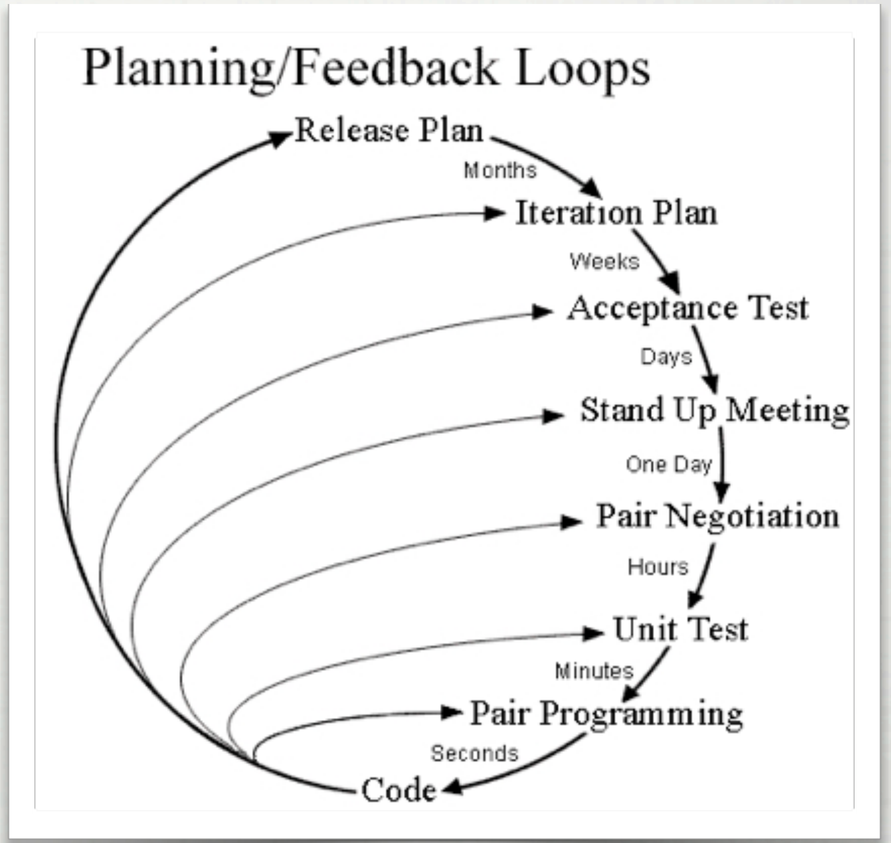
WATERFALL



- The Waterfall method was introduced in the 1970s
- One of the earliest software design life cycles (some evidence points to the Waterfall method being around even earlier, 1950s)
- Originally, the Waterfall method lacked feedback systems, this flow chart adds in feedback to all major portions of the process
- Modeled after the manufacturing and construction life cycles, where modifications late in the game are cost prohibitive (hence no feedback)
- Still in wide use

EXTREME PROGRAMMING

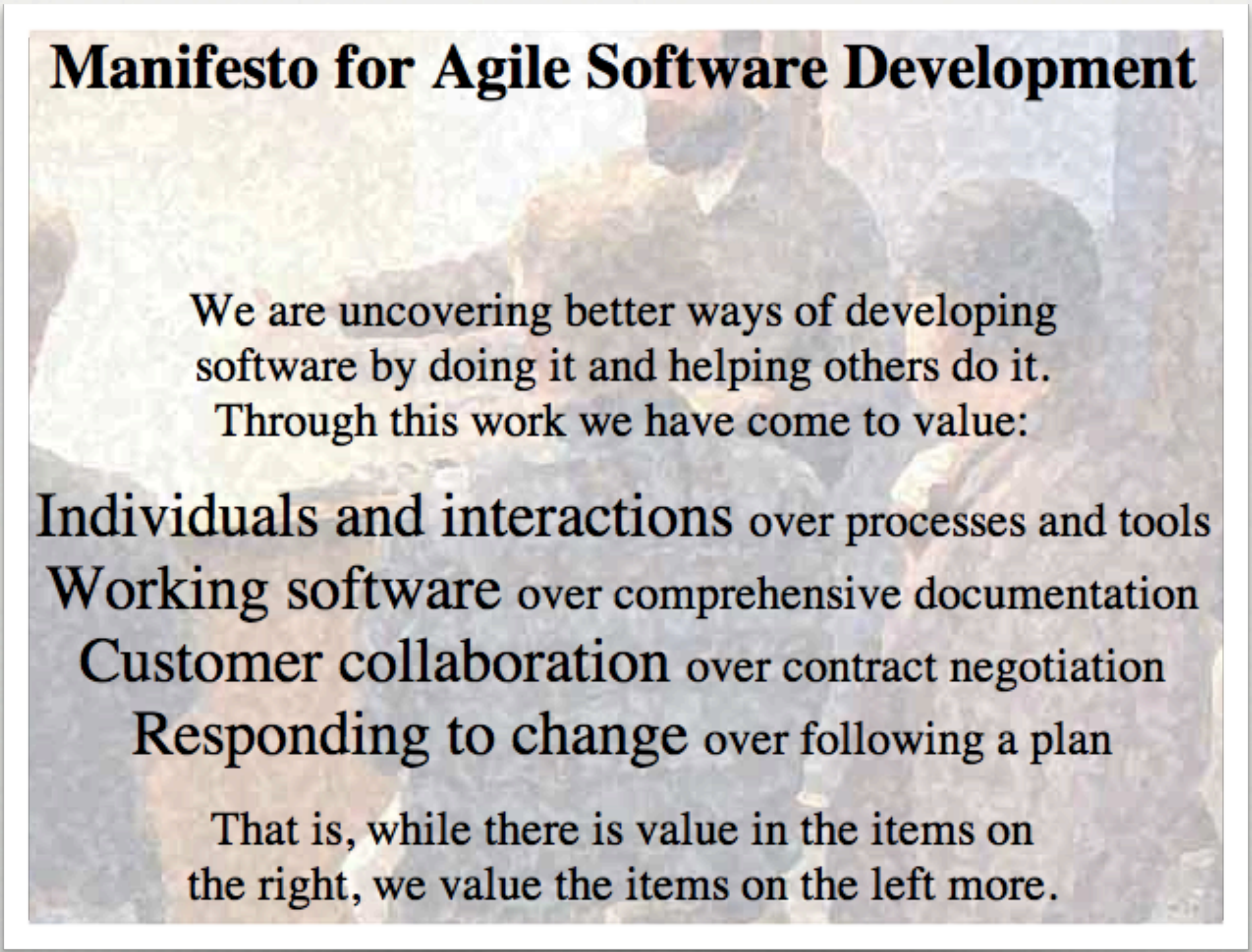
- Extreme programming is a type of 'Agile' approach (more in a minute)
- Extreme programming tries to incorporate as many iterations and feedback loops as possible in hopes of tuning the system as accurately as possible
- Created by Kent Beck in 1996
- Extreme programming employs some unique, yet effective programming techniques:
  - Pair Programming
  - Code Clarity
  - Flat Management
  - Many more



Agile

- Agile is almost a state of mind with which to approach software engineering and development
- It leaves behind the heavy document driven, ‘industrial strength’ life cycles of industry, and focuses on customer interactions and delivering software
- But not just delivering software, but delivering the RIGHT software by communicating with the customer
- Agile methods focus on a small, iterative approach to software development.
- Agile even goes so far as to encourage having a Customer as an active member of the design and development team
- Developed in 2001 by many leaders in the field of software engineering and development, Agile hopes to provide a new and effective approach
- Please see the next slide for a clipping from the Agile Manifesto Homepage

AGILE MANIFESTO

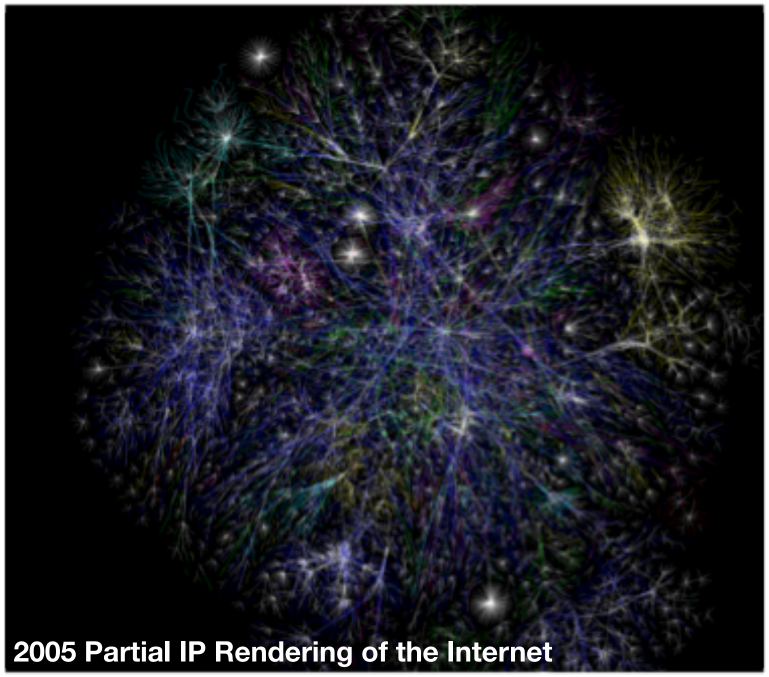

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

CLIPPING FROM THE AGILE MANIFESTO HOME PAGE, CLICK IMAGE FOR LINK

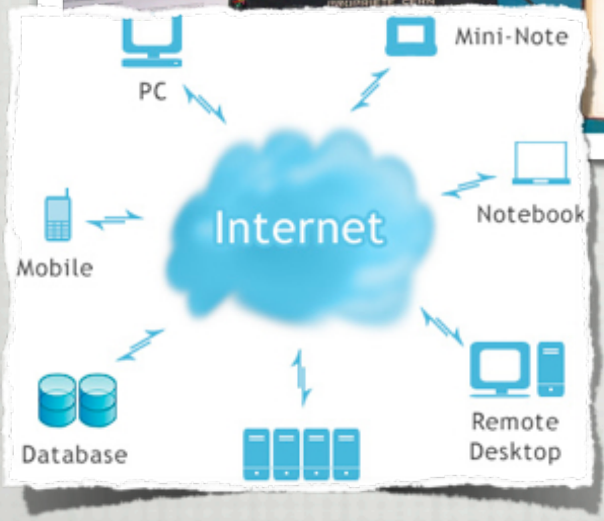


CHANGING ENVIRONMENTS

MICROCOMPUTERS AND THE PC

- First appearing in the 1970s, microcomputers put computers in the hands of hobbyists
- Finally, the ‘average man’ could afford a computer, which of course pushed software engineering and development, often from both ends
- The availability of hobby systems soon gave rise to enthusiasts and entrepreneurs (ie Steve Jobs, Bill Gates) who’s companies would at times drive and be driven by the software engineering industry

DISTRIBUTED SYSTEMS AND THE INTERNET



- The advent of the Internet and the personal computer revolution, further drove the Software Engineering profession
- With the interchange of media, gaming, cloud storage; people demanded (and continue to demand) capable software systems
- Distributed and multicore systems also drive the speed and complexity of new software engineering projects
- New methods of development are now possible, such as Follow-the-Sun ‘global’ workflows of program development.

Further Reading

SWEBOK	Online “Guide to The Software Engineering Body of Knowledge”
National Museum of Computing	Online resources for the British national Museum of Computing
IEEE Milestones	Online listing all of the official IEEE Milestones and supplemental text