Lecture 20
A Brief History of Computers & Programming Languages

See "The Innovators: How a Group of Hackers, Geniuses, and Geeks Created the Digital Revolution" by Walter Isaacson
Copyright © 2014 by Walter Isaacson
In the beginning… (1/12)

- Calculating and Calculus come from Latin word calculus (“a pebble or stone used for counting”), diminutive of calx (“limestone”).
- Crucial inventions are the positional numbering system, zero, and early algorithms such as multiplication and division.
  - Algorithm is named after 9th-century Persian mathematician and astronomer Al-Khwarizmi who was trained in Indian and Greek sources.
  - Algebra is derived from al-jabr, one of the operations he used for solving quadratic equations.
- Zero appears to have been invented by Babylonians, Mayans, and Chinese. Carbon dating of an ancient Indian document, the Bakhshali manuscript, has recently placed the first written occurrence of the number zero in the third or fourth century AD, about 500 years earlier than previously believed.

In the beginning… (2/12)

- Suanpan/Soroban/Abacus and other counting devices in multiple civilizations centuries AD! (and still in use!)
- Tables, logarithms, and slide rules.
  - 1614: Napier publishes logarithms.
  - 1620s: First slide rules based on logs, effectively analog computing.

In the beginning… (3/12)

- Antikythera mechanism
  - Oldest known example of an analog computer dated ~200 BC.
  - Discovered in a Greek shipwreck in 1901.
  - 37 bronze toothed gears enabling it to follow the movements of the Moon and the Sun through the zodiac, to predict eclipses, and to model the irregular orbit of the Moon.
Long history of mechanical clocks, orreries, elaborate music boxes and instruments, up to and including today's player pianos (listen to Gershwin)

Mechanized music museum in Utrecht, NL – Museum Speelklok (player clock, like player piano)

Museum dedicated to mechanical music making

- Rush E – The Impossible Piano Piece
- Wintergatan – Marble Machine

Blaise Pascal

Pascaline

Leibniz

Calculating Machine

1645: Blaise Pascal creates the Pascaline adding machine – up to 8 dials

1672: Leibniz Calculating Machine (repeated reckoner) could add, subtract, multiply, and divide

1801: Jacques Étienne Marie Leclercq gets award for pattern loom driven by punched cards

The Jacquard Loom

1725: Perforated paper roll to control the pattern woven on a loom by Bouchon

1728: Fabron linked punch cards together by string

1801: Jacquard gets award for pattern loom driven by punched cards
1822: Eccentric British inventor Charles Babbage proposed idea of mechanical calculation to compute polynomials (for ballistics, longitude tables): the Difference Engine
- The machine was designed but not built
- The name derives from its method of divided differences

A modern implementation of the Difference Engine was finally completed by the London Science Museum in 2002.

Proposed combining mechanical calculation with idea of feeding instructions via punched cards in the style of music boxes and the Jacquard loom, thus designing the first (mechanical) computer: the Analytical Engine
- First had to invent tools for the precise machining required, but the Analytical Engine was never completed
- However, the "architecture" is strikingly similar to essence of modern computers: driven by instructions, has arithmetic unit, memory, input/output

Babbage's son built a small part of his Analytic engine in 1910, and the Science Museum has begun the process of building a complete version.

1845: John Clark, cousin of founders of the Clarks' shoes empire, built the Eureka, a machine that produced "polished lines of Latin poetry"
- One of the forerunners of the programmable computer
- Machine that generated Latin hexameter verse
- Hexameter is the meter of ancient epic, of the poets Ovid and Virgil
- The strict rules of Latin hexameter make it similar to following a mathematical formula
- 26 million possible permutations
- "If we had it running continuously, it would take 74 years for it to do its full tour before it started repeating itself."
Augusta Ada Lovelace (Lord Byron’s daughter) is a mathematician, poet and philosopher, who wrote a program to calculate Bernoulli numbers for the Analytical Engine — the first known computer program and programmer!

- “The Analytical Engine weaves algebraic patterns the way the Jacquard Loom weaves flowers and leaves.”
- Ada programming language named in her honor was a DOD-sponsored language for writing programs using software engineering principles, including Abstract Data Types.

In the beginning… (9/12)

- 1836: Census bureau used Hollerith’s tabulator fed by punch-cards
- 1900s: Specialists programmed “business machines” (note IBM’s original name) by actually changing hardware’s wiring

In the beginning… (10/12)

- 1930s: The IBM Automatic Sequence Controlled Calculator (ASCC), called Mark I was designed by Harvard University’s staff.

In the beginning… (11/12)

- 1930s: The IBM Automatic Sequence Controlled Calculator (ASCC), called Mark I was designed by Harvard University’s staff.

First actual case of bug being found. See: “The moth in the relay”
In the beginning... (12/12)

  - First electronic general purpose “automatic” computer, used for ballistics and bomb calculations.
  - 18,000 vacuum tubes, MTBF of 20 minutes! Still programmed by wiring.
  - 5,000 adds/subtracts/sec, 400 multiplies/sec, 35 divisions or square roots/sec (10 digit numbers).
  - Men wanted to build hardware, left less prestigious “programming” to the all-female corps of programmers, called “computers,” a 19th century term.

Belatedly Recognized Female Pioneers (1/2)

- During World War II, a group of women were chosen for their abilities and math.
- They decoded several Japanese messages which enabled the US to sink military ships and stop enemy attacks.
- Among them was Ann Caracristi who became the first female NSA director in 1980.
- “The Women Who Helped America Crack Axis Code.”

Belatedly Recognized Female Pioneers (2/2)

- “Hidden Figures: The True Story of Four Black Women and the Space Race” by Margot Lee Shetterly, also made into a movie in 2016.
- Tells the story of the black women who worked in NASA, in segregated Virginia and contributed to the successful launch of the first man in space.
- Katherine Goble made accurate calculations which allowed the Friendship 7 shuttle to launch successfully.
- Dorothy Vaughan taught herself FORTRAN so she and her co-workers could operate the new IBM machines, and was named supervisor.
- Mary Jackson obtained her engineering degree and was hired as an engineer at NASA.

Belatedly Recognized Female Pioneers (3/2)

- Tells the story of the women who worked at Bell Labs and contributed to the development of computers in the 1950s.
- Grace Murray Hopper created the FIRST high-level programming language, COBOL, and was named to the National Women’s Hall of Fame.
- Grace Hopper also became the first woman to be promoted to the rank of Rear Admiral in the US Navy.

Belatedly Recognized Female Pioneers (4/2)

- Tells the story of the women who worked at the National Bureau of Standards and contributed to the development of computers in the 1960s.
- Betty Hoffa was the first woman to be named to the National Women’s Hall of Fame.
- Betty Hoffa also became the first woman to be promoted to the rank of Rear Admiral in the US Navy.

Belatedly Recognized Female Pioneers (5/2)

- Tells the story of the women who worked at the National Bureau of Standards and contributed to the development of computers in the 1960s.
- Betty Hoffa was the first woman to be named to the National Women’s Hall of Fame.
- Betty Hoffa also became the first woman to be promoted to the rank of Rear Admiral in the US Navy.

Belatedly Recognized Female Pioneers (6/2)

- Tells the story of the women who worked at the National Bureau of Standards and contributed to the development of computers in the 1960s.
- Betty Hoffa was the first woman to be named to the National Women’s Hall of Fame.
- Betty Hoffa also became the first woman to be promoted to the rank of Rear Admiral in the US Navy.

Belatedly Recognized Female Pioneers (7/2)

- Tells the story of the women who worked at the National Bureau of Standards and contributed to the development of computers in the 1960s.
- Betty Hoffa was the first woman to be named to the National Women’s Hall of Fame.
- Betty Hoffa also became the first woman to be promoted to the rank of Rear Admiral in the US Navy.

Belatedly Recognized Female Pioneers (8/2)

- Tells the story of the women who worked at the National Bureau of Standards and contributed to the development of computers in the 1960s.
- Betty Hoffa was the first woman to be named to the National Women’s Hall of Fame.
- Betty Hoffa also became the first woman to be promoted to the rank of Rear Admiral in the US Navy.

Belatedly Recognized Female Pioneers (9/2)

- Tells the story of the women who worked at the National Bureau of Standards and contributed to the development of computers in the 1960s.
- Betty Hoffa was the first woman to be named to the National Women’s Hall of Fame.
- Betty Hoffa also became the first woman to be promoted to the rank of Rear Admiral in the US Navy.
1945: John von Neumann publishes seminal concept of “stored program” computer, aka von Neumann architecture
- Wrote first paper detailing the idea, based on discussions with Mauchly’s team, influenced by Turing’s work on the Universal Turing Machine (1936)
- “it’s bits all the way down…” for both data and instructions
- Program can be stored in main memory and even treated as data to be operated on; paved the way for modern computers
- ENIAC+ embodied the idea (1948), UK Manchester’s Baby (1948) first design with von Neumann architecture

1957

Simple instruction execution loop in CPU
- Use instruction register to fetch instruction stored at that address in memory, increment instruction counter, decode and execute instruction
- Instruction typically is: op_code + memory address

Moore’s Law and the Shrinking Computer (1/2)
- Moore’s “Law”, an observation that over the history of computing hardware, number of transistors on a dense integrated circuit doubles approximately every two years
- In < 6 years it increases an order of magnitude!
- Smaller feature size, greater density, fewer shorter paths, faster signal propagation in microprocessors
- We benefit not just from miniaturization of the CPU but also from great electromechanical engineering of peripheral devices (e.g., disk controllers, disks)

CS15 Final Project Othello uses the minimax algorithm!
- Instruction may update memory – even modify stored program itself e.g., for loops, unsafe not to keep index registers

Moore’s Law and the Shrinking Computer (2/2)
- Gordon Moore was the co-founder of Intel and the co-inventor of the integrated circuit, which led to microprocessors, etc.

Stored Program Computer Architecture (1/2)

Stored Program Computer Architecture (2/2)

Moore’s Law and the Shrinking Computer (1/2)
Mainframe Machine Room

IBM System 360/65 mainframe with 2314 disk unit; 8x25MB = 200 Mbytes (late 1960's)

1100 x IBM 2316 29Mbyte disks in one 32 Gbyte microSD card

IBM Z13 microprocessor (6-8 cores @5 Ghz) has about same computing power as two football fields worth of Brown's IBM /360 mod 50's (.14MIPS) 50+ years ago — mainframes still selling!

A Look at Mainframe History as IBM System /360 Turns 50, COBOL Turns 55

But are Silicon chips hitting a limit? Transistors with a single atom?

What's next: biological computer? Quantum computer?

- Can store multiple states simultaneously, like fundamental and harmonic in sound, allowing parallel storage and computation
- Factorization for large N shown to be doable with quantum computers - would wreak havoc with encryption, e-commerce
- Making a full QC, let alone a general purpose QC using non-semiconductor solutions, still in research stage
- "arms race" between China and US for supremacy in QC

Moore's Law and the Shrinking Computer (2/2)

Moore's Law and the Shrinking Computer

Computers get faster, but do they get more "powerful"?

- Computer is the only Universal Machine!
  - Yet theoretically, only need 6 instructions for ANY algorithm:
    - load accumulator (high-speed register) from memory address
    - store accumulator to memory address
    - add the contents of memory address to accumulator, store result in accumulator
    - jump to memory address (branch) if accumulator < 0 (conditional jump)
    - read from accumulator from external input device
    - write from accumulator to external output device
  - You can build
    - add by subtracting negative numbers
    - divide by repeated subtract, multiply by repeated add
    - if then yes and loop with conditional jump
    - output to printer by write into special memory location

- IBM Z13 microprocessor (6-8 cores @5 Ghz) has about same computing power as two football fields worth of Brown's IBM /360 mod 50's (.14MIPS) 50+ years ago — mainframes still selling!

- A Look at Mainframe History as IBM System /360 Turns 50, COBOL Turns 55

- But are Silicon chips hitting a limit? Transistors with a single atom?

- What's next: biological computer? Quantum computer?

- Can store multiple states simultaneously, like fundamental and harmonic in sound, allowing parallel storage and computation

- Factorization for large N shown to be doable with quantum computers - would wreak havoc with encryption, e-commerce

- Making a full QC, let alone a general purpose QC using non-semiconductor solutions, still in research stage

- "arms race" between China and US for supremacy in QC

- Computer is the only Universal Machine!
  - Yet theoretically, only need 6 instructions for ANY algorithm:
    - load accumulator (high-speed register) from memory address
    - store accumulator to memory address
    - add the contents of memory address to accumulator, store result in accumulator
    - jump to memory address (branch) if accumulator < 0 (conditional jump)
    - read from accumulator from external input device
    - write from accumulator to external output device
  - You can build
    - add by subtracting negative numbers
    - divide by repeated subtract, multiply by repeated add
    - if then yes and loop with conditional jump
    - output to printer by write into special memory location
Tradeoffs in Power/Complexity of Instruction

- Tradeoffs:
  - Complexity of instruction (how much it does)
  - Speed of instruction execution
  - Size of instruction set (and can compiler take advantage of them)

- Today's computers:
  - Complex Instruction Set Computer (CISC) > 500
    ▪ Started with IBM mainframes in 50s and 60s, now "Intel architecture" dominates
  - Reduced Instruction Set Computers (RISC) 100–300 (simpler but faster instructions)
    ▪ Major innovation and important in 80s and 90s
    ▪ Intel architecture has adapted best ideas from RISC
    ▪ ARM architecture also RISC: used in phones, tablets, Macs, Arduino boards for IoT
  - Emphasis today is on "multi-core" (multiple CPUs per chip) and low-power designs
    ▪ GPUs (Graphics Processing Units) are even more powerful. For games, but also for data crunching, e.g., scientific simulation, weather prediction, protein folding, machine learning, LLMs… increasingly use GPU clusters for heavy duty computation

Turing, Computability (1/3)

- Alan Turing (1912–1954): logician, mathematician, cryptanalyst, first computer scientist, theoretical biologist
  - Designed code breaking machine "the Bombe" to crack the WWII German Enigma cypher – crucial in winning the war!
  - Envisioned a test for when to call a machine "intelligent" - Turing test

Turing, Computability (2/3)

- Formalized notions of algorithm, computer and mechanized computing in an era that was very concerned with what was computable and what was not, mechanized mathematics, e.g., under/over/missing, testing, problem, etc…. also started AI, Turing Test
  - Turing Machine as the simplest possible conceptual device to execute an arbitrary algorithm: device with a writable tape and a read/write head, the logic is in a table.
### Turing, Computability (3/3)

- Table contains the "program" of "instructions" as a "state machine"– if in state $i$ and read 1, do action $x$, then go to a next state; if read 0, do action $y$, go to a next state.
  
  Simple actions:
  1. move the r/w head one square left or right
  2. read/write current cell (empty or tape alphabet)
  3. does not have to halt

- Universal Turing Machine that could simulate any other TM by simulating its table.
  - proof that one could build a universal "programmable" computer
  - MIT's AI pioneer Marvin Minsky devised a 43-state UTM!

- Turing committed suicide after being prosecuted and "treated" (chemically castrated) for being gay
  - PM Gordon Brown publicly apologized in 2009, Queen Elizabeth granted a posthumous pardon in 2013

---

### First, Numeric Machine Language, Then Came Assembly Language (1/2)

- **1949**: John Mauchly develops Short Order Code
  - first assembly language
  - provided vehicle for higher-level languages to be developed

- Symbolic code that is 1:1 with machine code

<table>
<thead>
<tr>
<th>opcode</th>
<th>memory address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD 4</td>
<td>0001 0000 0000 0100</td>
</tr>
</tbody>
</table>

- Symbolic code that is 1:1 with machine code
  - load accumulator with contents stored at address 4
  - program translates to machine code via table lookup of opcode, decimal to binary conversion algorithm
  - assembler early example of treating a program as data!

- Must be defined for each processor
  - hardwired for particular processor's architecture
  - generated by compilers for higher-level languages

- Modern processors are very complicated
  - no writing at assembly language level takes real skill (learn it in CS33)
  - programmers generally optimize code only locally

- Still used today when speed and size count
  - embedded computers, device drivers, games
  - programmer must understand hardware well to use effectively
  - increasingly, C is used as a "machine-independent" assembly language (CS33)
High-Level Languages

- Attempt to make programming more intuitive
  - closer to programmer’s concepts (high-level)
  - farther from machine’s concepts (low-level)
- Symbolic code that is 1:N with machine code
  - one high-level instruction may become tens or hundreds of machine code instructions
- Most importantly, machine-independent
  - avoided vendor lock-in
  - dependent on compiler to translate high-level constructs to computer’s machine code
- Still trying to make languages higher level
  - Java guarantees single compilation, same execution on multiple machines via byte code: write once, run everywhere
  - compile to byte code virtual machine; computer will have virtual machine interpreter

High-Level Languages: Important Dates (1/2)

- 1957: John Backus et al. at IBM develop FORTRAN language and compiler
  - FORmula TRANslator
  - still used today, mostly for scientific computing, highly optimized for number crunching
- 1959: Committee on Data System Languages develops COBOL
  - led by Rear Admiral Grace Hopper, one of first modern programmers
  - Common Business Oriented Language, “English-like,” support for data records
  - still tons of legacy code in banks, insurance companies, retail… (Y2K!)

- 1959: John McCarthy develops LISP
  - LISt Processing
  - seen as slow, so primarily used only for “AI” projects
  - Scheme is a modern Lisp-like “functional programming” language
  - Python (C2000) can be seen as language with Lisp functionality, but with modern syntax
  - Pyret (C2011) is also a similar language, and often used for education
  - [http://norvig.com/python-lisp.html](http://norvig.com/python-lisp.html)
  - [Pyret (C2011)](https://www.pyret.org/pyret-code/)

- 1960: ALGOL 60 standard published
  - ALGOrithm Language; formally defined; task/procedure decomposition oriented
  - basis of most popular languages today
- 1964: John Kemeny and Thomas Kurtz at Dartmouth develop BASIC
  - Beginners All-purpose Symbolic Instruction Code
  - simple language, meant to be used by beginners and non-professionals, efficient on microcomputers
  - Visual Basic, now largely replaced by JavaScript and TypeScript (OOPLs)
Structured Programming (1/2)

- 1968: Edsger Dijkstra writes landmark note: "GoTo Statement Considered Harmful"
  - GoTo, an unconditional branch without a return, leads to tangled code
  - no predictability, GoTo can go anywhere in program, can't be understood by programmer or compiler

- New languages would have constructs for common one-in-one-out flows of control for controlled branching-the return from the branch is prescribed
  - if/else, if and switch statements
  - while and for loops

- Gives sequential, predictable order to code, only controlled branches allowed
- allows better code optimization

Brown's AM101, AM40, CS11 (CS15 precursors) switched to new structured programming style using only 1-in, 1-out branching in late 60's

Next Generation High-Level Procedural Languages

- Emphasize task decomposition, no bundling of data and procedures in "objects"

- 1964: Researchers at IBM develop PL/I, an omnibus language
  - Programming Language I
  - designed to synthesize best features of FORTRAN, COBOL, and Algol 60
  - initial attempt to be one general-purpose programming language

- 1970: Niklaus Wirth develops Pascal
  - named for Blaise Pascal, designed to be an educational language

- 1972: Dennis Ritchie at Bell Labs develops C (also learned in CS33)
  - predecessor named B
  - often called portable assembly language
  - surpassed COBOL as most popular language

Object-Oriented Programming Languages (1/3)

- Even OOPs are relatively new!

- 1967: Ole-Johan Dahl and Kristen Nygaard at Norwegian Computing Centre develop Simula, SIMUlation Language and first OO programming language, classes

- 1972: Alan Kay, Adele Goldberg, Dan Ingalls (all Computer History Fellows), et al at Xerox develop Smalltalk and the windows metaphor/GUI

- 1973: Barbara Liskov at MIT develops C++ with focus on ADTs (next slide)
Object-Oriented Programming Languages (2/3)

- 1980: US Department of Defense develops Ada to combat plethora of languages whose code doesn't interoperate
  - ADT's, Objects, Concurrency...
  - like PL/I, an omnibus, complex language
defense contractors had trouble finding qualified staff to write in it
- 1983: Bjarne Stroustrup develops C++
  - OOP extensions to popular C-language—named C++ as a play on the ++ operator (one better than C!)

Object-Oriented Programming Languages (3/3)

- 1995: James Gosling et al. at Sun Microsystems develop Java, a cleaned-up, smaller dialect of C++
  - meant to be internet and embedded device programming language
  - provide facilities for better reuse and safety
  - some professionals avoid it because it is seen as inefficient (use C++ or C instead)
  - Microsoft’s C# is a powerful Java-ish competitor, also Python, Ruby on Rails
  - JavaScript is NOT Java, and is mostly an OOPL

Important note: OOP is one of multiple programming paradigms, not a panacea.
Procedural and functional programming, and special purpose languages like MATLAB and MATHEMATICA, are tools with their own applicability, and anyone developing s/w needs to be multi-paradigm, multi-lingual.

Remember APIs?

- Application Program(ming) Interfaces
  - coined by former undergraduate Ira Cotton in 1968
- Think JavaDocs: Collection of method/function invocations with their parameters and returns, with brief descriptions of their functionality, error conditions, etc.
- NO implementation details — encapsulation!
Who “owns” APIs? (1/2)

- Oracle vs. Google reuse of APIs and Java code
- November 7, 2014: Computer Scientists Ask Supreme Court to Rule APIs Can’t Be Copyrighted
  - The Electronic Frontier Foundation (EFF) filed a brief with the Supreme Court, arguing on behalf of 77 computer scientists that the justices should review a disastrous appellate court decision finding that application programming interfaces (APIs) are copyrightable.
  - The decision, handed down by the U.S. Court of Appeals for the Federal Circuit in May, up-ended decades of settled legal precedent and industry practice.
  - Signatures to the amicus brief include five Turing Award winners, four National Medal of Technology winners, and numerous fellows of the Association for Computing Machinery, IEEE, and the American Academy of Arts and Sciences.
  - The list also includes designers of computer systems and programming languages such as AppleScript, AWK, C++, Haskell, IBM S/360, Java, JavaScript, Lotus 1-2-3, MS-DOS, Python, Smalltalk, TCP/IP, Unix, and Wiki.

Who “owns” APIs? (2/2)

- June 29th, 2015: Supreme Court refuses to rule on Court of Appeals ruling upholding Oracle’s ownership of Java APIs, the suit for copyright infringement against Google is ongoing, with Google using “fair use” doctrine.
  - May 2016: Jury ruled in Google’s favor, using “fair use” doctrine.
  - March 2018: After Oracle lost an appeal, Federal Appeals Court overruled the jury and said Google’s use of Java was not “fair use.” Oracle is back in trial court.
  - January 2019: Google filed another petition asking the Supreme Court to review Federal Circuit Decisions.
  - April 5, 2021: Supreme Court rules Google’s use of APIs was “fair use” but sidestepped whether APIs are copyrightable IP, though seemingly favoring the idea.

Reddit API Scandal & Protests

- Summer 2023: Reddit began charging for usage of its API July 1st, which had previously been free to use.
  - a brutal price for outside indie developers
  - popular 3rd party apps forced to shut down
  - Christian Selig, developer of Apollo, would have owed Reddit $20,000,000/year at the current pricing!
  - Article: Why disabled users joined the Reddit blackout

- 8000+ subreddits “went dark” in protest
  - supposed to be 2 days, lasted 2+ months.
  - sizable portion of the site

- Question: even if you can copyright an API, is it enforceable?

References:
Wikipedia, Electronic Frontier Foundation
Software Engineering (1/4)

- 1968: NATO Science Committee addresses "software crisis"
  - Hardware progressing rapidly, but not software
  - Software development seen mostly as craft with too much trial-and-error
  - Too little has changed – e.g., ACA website debacle!
    - (and RI's multiple failed roll-outs: DMV, DHS SNAP, …)

- 1975: Frederick Brooks writes landmark book "The Mythical Man-Month"
  - Says "no silver bullet," software is inherently complex
    - Most complex man-made systems
  - Complexity can be ameliorated but cannot be cured by higher-level languages
  - Adding people to project delays it ("9 women can't make a baby in a month")

Software Engineering (2/4)

- 1990s: Les Hatton develops "30-5-1" rule
  - From study of real commercial programs
  - Discovered 30 bugs per 1000 lines-untested code on average, then only 5 in well-tested code, and 1 bug still remaining after code is production
  - Rule held regardless of language, probably still true today!
    - All commercial s/w has day-one bugs, and for non-life-threatening s/w, we tolerate it
    - Under the guise of "early availability," vendors let the user community debug
      - Unacceptable for mission-critical s/w used in nuclear reactors, weapons, vehicles, medical apparatus, DFT and other banking apps, IRS s/w, etc.
Software Engineering (3/4)

- Sophisticated development and testing methodologies
  - CS17 and CS19 teach students to write tests that inform the implementation rather than write tests that are tailored to the implementation
  - goal is to cover both general and edge cases
  - formal verification (proving hw and sw correct) is in the ascendency again

Software Engineering (4/4)

- Libraries of reusable components
  - companies offer verified and common components
  - "plug-and-play" frameworks to connect trusted catalogue parts
  - OOP/D is a good paradigm to make this goal feasible (web for GUI widgets; Java in Java, and large-scale components: e.g., "Enterprise JavaBeans"; "IT Framework" used in CS122)
  - CS32: modern software engineering using Java

- Note: new languages and software engineering technologies (frameworks, IDEs, ...) will be subject, both in industry and in academia
  - e.g., Apple's Swift, positioned as successor to C and Objective-C, and Google's Go, designed to make common problems in app development easier to solve

Implications of Information Technology

- Computing/IT history isn't just about all the great strides we've made, and the fact that we're still in the dawn of this technological revolution!
  - Growth was driven by techno-optimism and the disruption paradigm, focus of Schumpeter's "creative destruction" economic theory of the effects of innovation
    - Steve Jobs: personal computing
    - Bill Gates: information at your fingertips
    - Now we recognize that there can be unforeseen, harmful consequences, disruption
      - job displacement because of automation (eliminating and creating new jobs, upskilling ...)
      - hacking of personal and corporate data
      - influencing elections vs fake news, difficulty easy to hack voting machines.
    - Instead of bringing us together via affinity groups and virtual communities, these technologies have...
      - created bubbles, hardening positions, amplifying the echo chamber, fueling conspiracy theories
      - spread of alternate facts and the post-truth world
      - fueled anti-ethnic/religious biases, "nationalism"
    - Cambridge Analytica, psychographic profiles - micro-targeting. "YOU are the product"

Be techno-optimists, help improve our world, but think about unintended consequences.
A Brief History Of CS15 (nee AM101/102, AM 51, CS11,...)

- 1965: Punched cards on the IBM 7070 mainframe
  - Machine language and assembly language, all of CS except data bases, AI and theory
  - From one credit to two credits/semester, then split into different courses
- 1966: IBM System/360 Model 50, punched cards
- 1968: IBM System/360 Model 67, used with timesharing past 1978
- 1978: IBM System/360 Model 158
- Late 60's to early 70's: Timesharing on dial-up (modems) typewriter and 80x20 alphanumeric terminals, but punched cards still used through 1980
- 1979: CS department created with 7 faculty in Kassar House
- 1985: DEC VAX 11/780 for departmental use, advanced courses and research
- 1988: Forty Apollo workstations in a ramped auditorium (pre-SunLab)
  - Brown was one of the 3 “Workstation Schools” (with MIT and CMU), first Apple ed partner
- 1990: Sunlab in the new CIT
  - Populated with 80 Sun Microsystems workstations
  - Used by Brown CS students until homebuilt PCs with Linux OS in late '90s

Aside: Barbara Liskov’s Talk at Brown 11/06/14

- Biography
  - member of the National Academy of Engineering and the National Academy of Sciences, the National Academy of Inventors
  - ACM Turing Award (the Nobel prize of CS), IEEE Von Neumann medal, Brown Distinguished Professor
  - The Power of Abstraction
    - abstraction is at the center of much work in Computer Science
    - finding the right abstraction for a system as well as finding an effective design for a system implementation
    - abstraction is the basis for program construction, allowing programs to be built in a modular fashion
  - What I learned from her talk
    - ADTs need to describe the behavior, not just the method signatures, return types, errors: “pragmatics”
    - Java and other OOPs use and provide support for eliminating that complexity with other signatures
    - Java and other OOPs use and provide support for eliminating that complexity with other signatures
    - CS15 has de-emphasized inheritance, pushing interfaces and composition