Forwards or Backwards

https://forwardorbackwards.com



Why the World Still Runs on COBOL

Watch on YouTube: https://www.youtube.com/watch?v=_U288BVzeAY

COBOL, the Common Business-Oriented Language, was built in 1959 by a committee that met at the Pentagon, and first ran in 1960. Sixty-six years later it is still running much of the world's banking, insurance, payroll and tax infrastructure.

A widely cited estimate, first reported by Reuters in 2015 and still repeated across industry coverage in 2026, puts 220 billion lines of COBOL in active use, running 43% of the world's banking systems and 95% of ATM transactions. A separate, earlier Gartner estimate from 1997 put the figure at 200 billion lines running roughly 80% of business transactions.

COBOL's two-digit year storage was a major contributor to the Millennium Bug: some studies attribute as much as 24% of total worldwide Y2K remediation cost to COBOL systems specifically. The language survived Y2K; the workforce that maintains it has not kept pace, with industry surveys

placing the average COBOL programmer's age above 55 and estimates suggesting roughly three-quarters of the current workforce could retire within a decade. IBM says it has trained over 180,000 developers in COBOL through more than a dozen years of fellowship and training programmes.

Full replacement remains rare because rewriting undocumented decades-old business rules risks getting real payments wrong. The US Internal Revenue Service announced a Digital First Initiative in 2024 to migrate its COBOL tax-processing systems to Java over several years - the pattern most institutions follow: coexistence and gradual migration, not a single rewrite.

Chapters

1. A language older than the people who maintain it
2. One meeting, one Pentagon, one language
3. Grace Hopper's head start
4. Built for money, not for speed
5. The number that will not go away
6. Where it actually runs, in 2026
7. The bug that proved how fragile it all was
8. The workforce cliff
9. Why replace it, and why nobody fully has
10. Forwards, not backwards

1. A language older than the people who maintain it



Somewhere behind almost every card swipe, almost every pension payment and almost every tax return processed today, a piece of code written in nineteen fifty-nine is still doing the arithmetic.

The language is called COBOL, short for Common Business-Oriented Language. It is sixty-six years old this year.

Nobody planned for that. COBOL was built to solve one problem, for one decade. Instead it became the language that banks, insurers, airlines and governments still trust with the numbers that matter most, and nobody has found a way to fully retire it.

This video is about why a language most programmers have never touched still sits underneath so much of the world's money, and why replacing it keeps turning out to be harder than it sounds.

2. One meeting, one Pentagon, one language



The story starts on the eighth of April nineteen fifty-nine, when a computer scientist named Mary Hawes organised a meeting at the University of Pennsylvania. Businesses across America were buying computers, and every manufacturer's machine spoke its own private language. A program written for one computer could not run on another.

That meeting led to a bigger one. [CALLOUT] On the twenty-eighth and twenty-ninth of May nineteen fifty-nine, forty-one people met at the Pentagon, chaired by Charles Phillips of the United States Department of Defense. The Department had hundreds of computers on order, from different manufacturers, and no common language to program them in. A week later, on the fourth of June, the group that came out of that meeting formally organised itself as CODASYL, the Committee on Data Systems Languages.

Their goal was specific: a language for business, not for scientists or engineers, that read close enough to plain English for a manager to check it, and that would run on any manufacturer's machine without being rewritten.

3. Grace Hopper's head start



CODASYL did not start from nothing.

A US Navy rear admiral and computer scientist named Grace Hopper had already built a language called FLOW-MATIC, the first to let programmers write instructions in English words rather than mathematical notation. Hopper served as a technical adviser to the committee, and in nineteen eighty she said plainly that COBOL sixty was ninety-five per cent FLOW-MATIC. One of COBOL's own lead designers, Jean Sammet, was careful to say Hopper was not COBOL's inventor. What is not disputed is that without FLOW-MATIC to borrow from, the committee would have had far less to build on.

The first COBOL program ran on the seventeenth of August nineteen sixty, on an RCA 501. Four months later, on the sixth and seventh of December, the same program ran unchanged on both an RCA and a Remington-Rand UNIVAC computer. That was the entire point proven in one demonstration: one language, many machines.

4. Built for money, not for speed



COBOL was never designed to be fast, or clever, or small. It was designed to be read.

Its instructions look like sentences: ADD, SUBTRACT, MULTIPLY, MOVE, COMPUTE. A manager without a programming background could follow the logic of a COBOL payroll program line by line, which is exactly what the committee wanted for a language that would be handling other people's money.

It also did something quieter that turned out to matter enormously: it handled decimal numbers exactly. Currency has to add up to the penny, every time, and COBOL's packed-decimal arithmetic does that natively, without the small rounding errors that binary floating-point math can introduce. Fifty years later, that one design choice is still a reason banks are cautious about moving off it.

By the nineteen seventies, COBOL was the dominant language for business computing in the world.

5. The number that will not go away



In nineteen ninety-seven, the research firm Gartner made an estimate. About two hundred billion lines of COBOL code were in active use. They said that code ran roughly eighty per cent of the world's business transactions.

A different estimate is more widely repeated today. Reuters reported it in twenty fifteen, and the industry still cites it a decade later. It puts the figure at two hundred and twenty billion lines of active COBOL code. That code was said to run forty-three per cent of the world's banking systems. It was also said to run ninety-five per cent of the transactions passing through automated teller machines.

Both figures are estimates, not counts, because nobody has ever audited every mainframe on Earth. But the direction they point in has not changed: this is not a language quietly disappearing. It is a language quietly still carrying an enormous share of the world's financial plumbing.

6. Where it actually runs, in 2026



The applications are not exotic. Core banking systems that post deposits and withdrawals. Insurance policy administration. Airline reservation systems. Payroll and pension calculations for government employees. Tax processing.

The United States Internal Revenue Service is a case study on its own. Its main tax processing systems have run on COBOL for decades. In twenty twenty-four the agency announced a Digital First Initiative to move that processing from COBOL to Java, a project expected to take years, not months, precisely because of how much depends on the old code behaving exactly as it always has.

That is the pattern almost everywhere COBOL still runs: not a system nobody dares touch, but a system nobody can afford to get wrong while touching it.

7. The bug that proved how fragile it all was

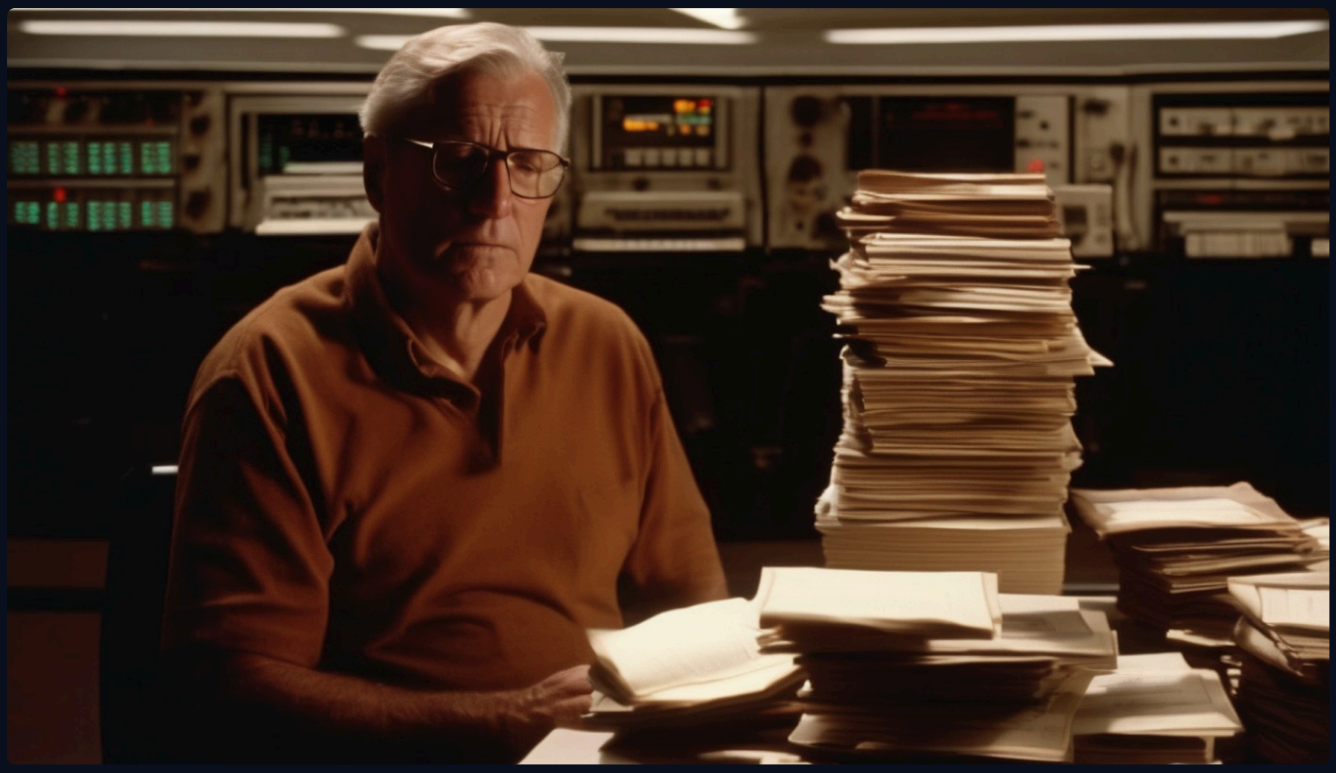


For most of COBOL's life, programmers stored years as two digits to save expensive storage space: nineteen ninety-nine was simply "99". As the year two thousand approached, that shortcut became a problem. A computer reading "99" could not tell nineteen ninety-nine from some other year ending in ninety-nine. Left uncorrected, it threatened to treat the year two thousand as nineteen hundred.

Some studies attributed as much as twenty-four per cent of the total worldwide cost of fixing the "Millennium Bug" specifically to COBOL systems, because so much of the affected code, in finance and government above all, was written in it.

The fix worked. The systems kept running past midnight on the thirty-first of December nineteen ninety-nine. But Y2K also proved something that has shaped every decision about COBOL since: when code this old and this deeply embedded goes wrong, the cost of finding out is measured in the running of the world's banks, not in a support ticket.

8. The workforce cliff

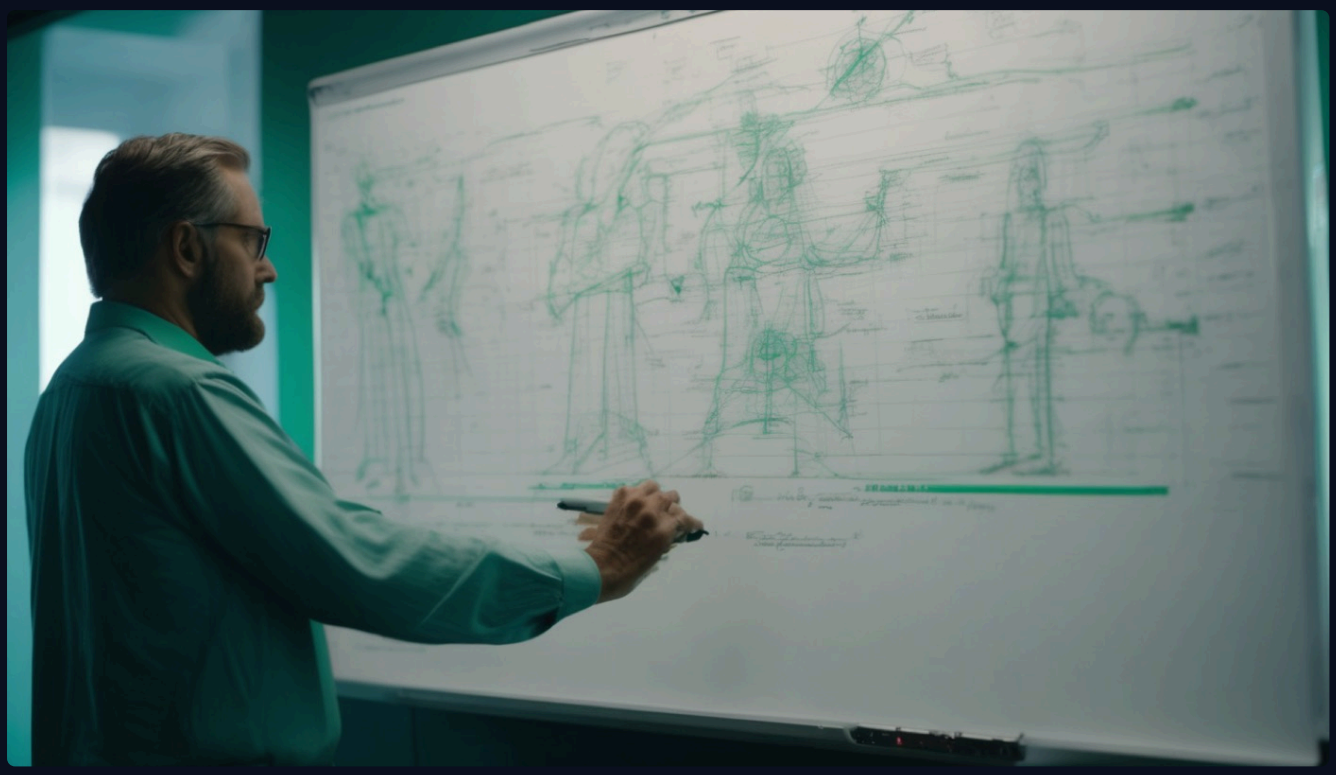


The language survived Y2K. The people who know it are not surviving in the same numbers.

Industry surveys have placed the average age of a working COBOL programmer above fifty-five, and estimates in the trade press suggest roughly three-quarters of the current COBOL workforce could retire within a decade. IBM has responded by training more than one hundred and eighty thousand developers in COBOL through fellowship and training programmes run over more than a dozen years, aimed directly at that gap.

The scarcity has a price. Experienced COBOL specialists routinely earn salaries and contract rates well above the market average for other programming languages, simply because so few people can maintain what is running.

9. Why replace it, and why nobody fully has



Every large organisation running COBOL has, at some point, considered replacing it. Almost none have replaced all of it.

Rewriting a core banking or tax system means re-implementing decades of business rules that were never written down anywhere except in the code itself, and testing the replacement against every edge case the old system has quietly handled correctly since the nineteen eighties. A rewrite that gets even a small percentage of those rules wrong can miscalculate real payments to real people.

So the more common path is not replacement. It is coexistence. Newer languages get built around the COBOL core rather than instead of it. Translation tools convert COBOL logic into Java or another modern language. Increasingly, AI-assisted tools help a shrinking pool of specialists read and modify code that was never meant to still be running sixty-six years later.

10. Forwards, not backwards



COBOL is not a relic sitting untouched in a museum. It is live code, still processing real transactions today, still being patched, still being trained on, still being replaced one careful piece at a time rather than all at once.

That is not a story about a language stuck in the past. It is a story about an industry making one decision, project by project. Keep the plumbing that has not failed running for as long as it keeps not failing. Change it only where the cost of leaving it alone has grown larger than the risk of touching it.

Sixty-six years on, that calculation is still coming out the same way, for institutions that quite literally cannot afford to get it wrong.

Sources and credits

Primary sources

- Wikipedia, 'COBOL' - 8 April 1959 Mary Hawes meeting at University of Pennsylvania; 28-29 May 1959 Pentagon meeting, 41 attendees, chaired by Charles A. Phillips (DoD); CODASYL formed 4 June 1959; Grace Hopper's FLOW-MATIC influence and her 1980 '95% FLOW-MATIC' remark; Jean Sammet on Hopper not being COBOL's inventor; first COBOL program run 17 August 1960 on an RCA 501; 6-7 December 1960 cross-platform demonstration on RCA and Remington-Rand UNIVAC machines; 1997 Gartner estimate of 200 billion lines running 80% of business programs; up to 24% of Y2K repair cost attributed to COBOL in some studies; IRS Digital First Initiative announced 2024 to move off COBOL to Java.
- IEEE Spectrum, 'Long-Enduring COBOL May Still Have a Shelf Life', published 24 February 2022 - cites a Reuters 2015 estimate of 220 billion lines of COBOL in production, 43% of the world's banking systems, and 95% of the world's ATM transactions; notes the estimate's age and continued wide use in industry commentary.
- DataField.Dev, 'Is COBOL Still Used in 2026? Why This 65-Year-Old Language Still Matters', published 22 January 2026 - repeats the same core figures (220bn lines, 95% of ATM transactions, 43% of banking systems); reports industry estimates that roughly 75% of the existing COBOL workforce may retire within a decade and that surveys place the average COBOL programmer's age above 55; no primary source given for the workforce figures, presented in the script as industry estimates accordingly.

Not regulated financial advice.