Neel Shah
HIST-325 Final Paper

COBOL Cripples the Mind!: Academia and the Alienation of Data Processing

In early April 2020, deluged with over 200,000 unemployment insurance claims, the state of New Jersey's application website crashed. New Jersey was not alone. With thousands freshly laid-off in the midst of a global pandemic, states like Colorado and Rhode Island also reported large-scale system failures. The culprit, according to New Jersey governor Phil Murphy, was the sixty-year-old programming language COBOL. "Literally, we have systems that are 40 years-plus old, and there'll be lots of post-mortems," he said. "And one of them on our list will be how did we get here where we literally needed COBOL programmers?"[1] News coverage of the crash was also unsympathetic to COBOL, variously describing it as 'old', 'outdated', and 'dead.'

But COBOL is anything but a dead language. Designed to be a portable, readable and standardized business-purpose language, COBOL took the business world by storm when it was first released in 1960. By 1970, it was the most widely used programming language in the world.[2] Much of today's critical business infrastructure is still COBOL-based—95% of all ATM transactions use COBOL, and 500 million mobile phone users are connected by it every.[3] COBOL systems have proven to be resilient, efficient and reliable. In the case of New Jersey, as it turns out, COBOL wasn't to blame at all. The failure was in the Java-based frontend, meaning that claimants were hitting a wall before their application ever touched a COBOL system.

The continued reliance on COBOL is reason enough for its study as a technological artifact. But as the language itself nears the (human) retirement age, a close examination of its origins and reception provide a glimpse into a different era of computing. As Michael Mahoney puts it in his classic essay *What Makes the History of Software Hard and Why It Matters*[4] , "the history of software is the history of how various communities of practitioners have put their portion of the world into the computer." Replace 'software' with 'programming languages' and this logic holds—programming languages define a set of instructions for the machine, invariably representing the priorities ('worlds') of those who design and implement them. A history of COBOL, by this token, is

[1] *As Unemployment Claims Spike New Jersey Seeks COBOL Coders*, NJ Gov.
[2] Hicks, *Built to Last*.
[3] *Academia Needs More Support to Tackle the IT Skills Gap*, Microfocus.
[4] Mahoney, 1.

not just a list of its technical hits and misses. It is a history of the business community of the 50s and 60s that created it, the defense industry that supported it, and the programmers that received and used it.

To study an artifact, however, is not just to study its well-wishers. Any history of COBOL is incomplete without its detractors and discontents, of which there are many. Governor Phil Murphy was not the first to criticize (or in his case, scapegoat) COBOL. Many programmers, academics, and 'hackers' expressed their dissatisfaction with COBOL over the years, and for different reasons. Consider this acerbic entry for COBOL in the *New Hacker's Dictionary* (1996):

> "COBOL /koh'bol/ n. [COmmon Business-Oriented Language] (Synonymous with evil.) A weak, verbose, and flabby language used by card wallopers to do boring mindless things on dinosaur mainframes. Hackers believe that all COBOL programmers are suits or code grinders, and no self-respecting hacker will ever admit to having learned the language. Its very name is seldom uttered without ritual expressions of disgust or horror."[5]

Criticisms of COBOL, as the example above shows, highlight the different value systems held by different 'communities of practitioners'. Here, COBOL is clearly the antithesis of 'hacker' culture, revealing something about the community through this negation. The community I am interested in, however, is not hackers, but academia. As I will argue, the academic reception of COBOL was largely negative, and remains a blindspot in its historiography. This critical reception, I propose, can be read as a familiar battle over technological and aesthetic *standards*, now waged by academia and 'business' in the nascent field of data-processing. This 'battle' itself was a product of and highlights the changing nature of the defence-academy-business connection in the computing world of the 1950s and 60s.

[5] ES Raymond, 104.

COBOL IS BORN: PORTABLE, READABLE, CONTROLLABLE

In June 1978, the first ACM SIGPLAN History of Programming Languages (HoPL) conference was held in Los Angeles. The conference featured presentations on about a dozen languages, each of which was later published as a peer-reviewed paper in the *SIGPLAN Notices*. One of the languages presented at HoPL 1 was the now-teenaged COBOL, already the most widely-used programming language in the world. Jean Sammet—widely regarded as the 'mother' of COBOL for her leading role in its development—delivered her presentation[6] to a room full of computing pioneers, in a rare history straight from the proverbial horse's mouth. In painstaking detail, Sammet described meetings, funding sources, committee structures and motivations. Her account of COBOL is the primary source for the brief history I describe below.

COBOL's story began with Mary Hawes, a programmer at the Burroughs Corporation. In early 1959, Hawes had requested a "formal meeting involving both users and manufacturers . . . to develop the specifications for a common business language."[7] Hawes got her wish. In April 1959, a small group including Sammet and Grace Hopper met at the University of Pennsylvania's Computing Center to flesh out the objectives for a first formal meeting. Through Charles Phillips, they decided to ask the Department of Defense to sponsor this effort. The DoD responded enthusiastically, immediately convening a meeting at the Pentagon for May 1959. Phillips summarized the DoD reaction as follows:

> "The Department of Defense was pleased to undertake this project; in fact, we were embarrassed that the idea for such a common language had not had its origin by that time in Defense since we would benefit so greatly from the success of such a project."[8]

About forty people attended the Pentagon meeting. Among them were 15 representatives from manufacturers such as Honeywell, GE and IBM; representatives from government; users and consultants; and one sole representative affiliated with a university. This was Saul Goren, of the University of Pennsylvania.[9] He attended only one of the two conference days, and had little to do

[6] In Wexelblat, the COBOL Paper is 199 onwards
[7] Wexelblat, 200.
[8] Phillips, 1959b, n.p.
[9] Wexelblat, 240.

with COBOL's development afterwards. From the very start, COBOL's development and design was completely detached from academia, in a physical as well as an intellectual sense.

There were two key outcomes of the Pentagon meeting. The first was the agreement that a common programming language *should* be created, with some of its core features decided upon. After much debate over two days, these rudimentary features were laid out: it should employ "maximum use of simple English language,"[10] and it should be "easier to use, even if somewhat less powerful." There was also a recognition of "need to broaden the base of those who can state problems to computers."[11] At the heart of COBOL's conception, then, was a certain democratizing impulse. This was reflected in COBOL's English-looking design, meant to make code easier to read and write. Consider the following example that Mar Hicks cites[12], of a line of code that computes a social-security payment rounded to the penny. In FORTRAN, this would look something like: `TOTAL = REAL(NINT(EARN * TAX * 100.0))/100.0` while the corresponding COBOL code would read: `MULTIPLY EARNINGS BY TAXRATE GIVING SOCIAL-SECUR ROUNDED.` Thus a key feature of COBOL was to be its readability, a decision taken at its very first formal meeting.

Readability, but for whom? In a paper published in the IEEE's *Annals*, Ben Allen views COBOL as a socially-constructed piece of technology, arguing that "its English-like appearance was a rhetorical move designed to make the concept of code more legible to non-programming communities."[13] Within industry (the intended audience of COBOL), Allen argues, some parties stood to benefit from COBOL more than others. Programmers already understood the code that was being written within large businesses, but there was an important group to whom code was a total black-box: managers, who hitherto did not have the technical capabilities to understand the programming process. With the promise of English-looking code, management could dream of understanding and therefore controlling programmers, making them particularly invested in COBOL's creation. These social factors, Allen argues, enabled COBOL's success, shaping it into the

[10] Wexelblat, 201.
[11] ibid.
[12] Hicks, 2.
[13] Allen, 17.

verbose, English-looking language it would become. Allen's argument is debatable, but highlights an important qualification that needs to be remembered while discussing COBOL's ostensibly democratizing impulse. COBOL was meant to be readable, yes, but that also made it and the programming process more controllable by management.

Another key feature—and COBOL's *raison d'être*—was its *portability*. This was so obvious to the attendees at the Pentagon meeting that it went virtually undebated. After all, portability was what had driven Mary Hawes to kickstart this entire process. At the time, major computer manufacturers from IBM to Remington-RAND were racing to develop proprietary languages for their machines, which would run poorly (or not at all) on a competitor's hardware. In December of 1960, that changed forever. Sammet and team ran "essentially the same COBOL program"[14] on both RCA and Remington-RAND Univac computers. It worked like a charm. In Sammet's own words, "the significance of this lay in the demonstration that compatibility could really be achieved." COBOL was not only readable, but was also portable—an immense technological achievement that would shape industry for years.

Rome was not built in a day, and neither was COBOL. With the key features of readability and portability laid out, the second key outcome of the Pentagon meeting was the creation of a plan of action. The urgency of the situation was recognized—with proprietary languages quickly being developed across the board, the COBOL task force was divided into two parts. The first was the Short-Range Committee, whose mission was to create an interim, stop-gap language to nip proprietary language development in the bud. This was to be done by combining features of Grace Hopper's FLOW-MATIC, the Air Force's AIMACO and IBM's COMTRAN, three languages that loosely resembled the COBOL bill. The Short-Range Committee moved quickly, and by December of 1959 the fundamental concepts, structure and layout of COBOL had been established. By the end of 1960, COBOL 60 was produced and released. Even at this very early stage, COBOL had its skeptics and

[14] Wexelblat, 216.

doubters. As a practical joke, committee member Howard Bromberg even had a COBOL tombstone made[15] for his teammates (see above). But COBOL was anything but dead on arrival. The readable, portable, and potentially controllable language would go on to become the most popular programming language in the world.

It is important to note that despite the HoPL conference's stated aim of considering the "technical factors which influenced the development of certain selected programming languages,"[16] the histories written there were anything but purely technical. As Sammet's account and the analyses of Hicks and Allen demonstrate, the creation of a programming language is not done in a social vacuum. COBOL's creation was was a socially constructed process—if anything, it was more shaped by the whims of Honeywell's managerial elite[17] than by notions of technical efficiency. To paraphrase Michael Mahoney (who himself may have been quoting Fred Brooks), the history of COBOL is "only accidentally about computers."[18] In the following sections I will discuss the nature of the academy-business-defense connection and the academic reaction to COBOL, but this brief history of COBOL's creation should serve to highlight its social construction and uniqueness at the time as a readable, portable, business-oriented language.

COMMUNITIES OF COMPUTING IN A CHANGING WORLD

To understand the relationship between academia and COBOL, it is necessary to set the stage by briefly describing the historically unique nature of the academy-defense-business connection in the computing world. As I will argue, the end of the Second World War was the beginning of a divergence in priorities of academia and business, with the latter shifting its focus towards data-processing tasks for clients in industry.

While the academy and defense parts of the triangle are fairly self-explanatory, it is worth defining who and what falls under the umbrella term 'business' that I will continue to use. By

[15] COBOL Tombstone, *Computer History Museum*.
[16] Wexelblat, xvii.
[17] Wexelblat, 215.
[18] Mahoney, 4.

'business' I refer to early commercial machine manufacturers such as IBM, Honeywell, Burroughs and Remington-Rand that themselves have rich histories predating the computer. IBM serves as an illustrative example. Founded in 1896 under the name Tabulating Machine Company, IBM and many of its peers had been involved in data-processing long before the twentieth century enabled electronic, computerized data processing.[19] IBM in particular was founded by Herman Hollerith, the patent holder of punched cards. His firm grew rapidly during the interwar years—as the need for data processing rose sharply, sales of machines like tabulators and sorters which could effectively summarize punched-card information grew in military and commercial use. By 1922, IBM's revenues totaled $10.7 million, up from just $4.2 million in 1914. As James Cortada—lifelong employee and historian of IBM—notes, "IBM typifies the pattern of behavior within the equipment industry during this era."[20] Although IBM monopolized the punched-card industry, other firms were carving out similar niches for themselves. Burroughs made headway in typewriters, while NCR built cash registers, for example. The mainframe-builders of the fifties and sixties did not come up overnight—this 'business' community had historical ties to electronics and data-processing.

But it was the Second World War that really built the business-academy connection. During wartime, electronics and office-equipment manufacturers turned their attention to war-related projects. IBM, for example, turned to weapons manufacturing, while other 'businesses' helped develop radar and other sophisticated electronics.[21] Since much of military research was done on university campuses, the connections between defense, business and academy began to grow. The development of the ENIAC in 1943 is representative of this trend—designed to calculate artillery firing tables by the Army, it was formally dedicated at the University of Pennsylvania and took input from IBM card readers.[22] This three-way confluence is somewhat unique in technological history, and itself is a reflection of the military-centric development of modern computing.

By the end of the war in the 50s and 60s, the demand for this new digital computing was ever-growing, and now had a new source: commercial customers. While railroads, insurance

[19] Cortada, *Information Technology as Business History,* 48.
[20] ibid.
[21] Cortada, *Information Technology as Business History,* 49.
[22] *ENIAC at Penn Engineering*, Penn Engineering.

companies and the likes had always demanded data-processing services, newer services like airline companies opened avenues for 'businesses'. Some statistics encapsulate this meteoric growth: while the data-processing industry was worth less than $1 billion in the mid-50s, it stood at over $40 billion by the end of the 1970s.[23] The shift was not just a question of scale, however. The *function* of computers was also changing, from scientific uses to more classical accounting, payroll and inventory tasks. This itself was a reflection of broader commercial interest in these machines growing rapidly in the tumultuous 1950s. In 1953, some historians argue, all computer development activity was for the government.[24] By the end of the fifties this statement was patently false, with COBOL itself being a perfect counterexample.

Thus the Second World War pushed defense, academia and 'business' into close quarters, as they collaborated on military problems and hardware development. The end of the war, however, coincided with a boom in commercial demand for data-processing which increasingly became the priority of 'businesses' such as IBM. COBOL was born against this backdrop of steady divergence between academy and business—while the former remained occupied by scientific and theoretical problems, the latter was turning (or returning) to data-processing problems, now in a digital world.

ACADEMICS AGAINST COBOL: A BATTLE OVER STANDARDS

"The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence," wrote Turing Award winner Edsger Dijkstra in 1975.[25] In a memo entitled *How Do We Tell Truths that Might Hurt?,* Dijkstra railed against the popular programming languages of the day—even FORTRAN ("the infantile disorder") was not spared from criticism. But Dijkstra was not the only computer scientist averse to COBOL. The phenomenon was so widespread that in 1984, the September issue of *Computerworld* magazine published an article entitled *An interview: COBOL defender* with Donald Nelson, a long-time programmer. COBOL defenders were novelty, to hear Nelson tell it. "Lots and lots of computer science graduates are being churned out every day,

[23] Cortada, 53.
[24] ibid, 53.
[25] Dijkstra, *How do we tell truths that might hurt?*, 3.

and nearly every one of those graduates has had 'hate COBOL' drilled in to him," Nelson said.[26] In this section I will attempt to understand what seems to be a widespread aversion to COBOL in academia. As I will argue, much of this stemmed from a view of data-processing as 'simple' and inelegant, as opposed to more 'complex' and theoretically grounded computer science. This has its roots in the academia-business drift described in the section above, and can be read as a battle over technological *standards*.

The first (and only) author to systematically study this reaction was Ben Shneiderman, himself a distinguished computer scientist. In a 1985 paper entitled *The Relationship Between COBOL and Computer Science,* he attempted to offer "historical, technical and social/psychological perspectives on the fragile relationship between COBOL and computer science."[27] He opens the paper provocatively: "for a computer scientist to write sympathetically about COBOL is an act bordering on heresy," he says. The development of COBOL, Shneiderman argues, was intellectually separate from academia. No academics worked on the design team, there were no citations of academic work (a proxy for flow of ideas), and COBOL did not use the Backus-Naur form as a metasyntax. Besides these historical factors, Shneiderman also stresses social ones. "The rejection of COBOL," he says, "is a product of their (computer scientists') desire to avoid the business data processing domain."[28] This may explain why several programming language textbooks like MacLennan's 1983 bestseller[29] do not so much as mention COBOL. In his interviews, Shneiderman found a degree of snobbishness among the academic community, with some deriding the "trade school nature" of COBOL while others remarked that they were "hostile to teaching what is used commercially."[30]

It is important to note that COBOL was not without its legitimate technical criticisms. As Allen notes, COBOL's verbosity and attempt to mimic English made writing and debugging code difficult.[31] COBOL statements ended with a period, which was easy to misplace when writing

[26] Computerworld, ID/29.
[27] Shneiderman, *The Relationship Between COBOL and Computer Science,* 348.
[28] Shneiderman, 350.
[29] MacLennan, xi.
[30] Shneiderman, 351.
[31] Allen, 24.

nested IF statements. The attempt at imitating English was also over-the-top at times—COBOL reserved a large number of meaningless 'noise words' like "ARE", "WHEN" and "AT", which were discarded prior to compilation and could not be used as variable names. A serious defect of COBOL was its inability to define functions with local variables; the original COBOL 60 only allowed for global variables. This made generic subroutines like summing the elements of an array difficult to write.[32]

The presence of these legitimate criticisms are what make the continued focus on the 'simple', data-processing nature of COBOL all the more surprising. Consider Terence Pratt's textbook *Programming Languages: Design and Implementation* from 1984, which dismisses COBOL's "orientation towards business data processing" and its algorithms as "relatively simple."[33]At the HoPL I conference, Jean Sammet herself pushed back against this notion. "I'm sure this will step on a great many toes," she said in the question-answer session, "[but] I think simply the data-processing activity is much harder . . . it's always been a very great source of sadness to me that so many people who clearly have the intellectual capability to deal with these kinds of problems on an abstract basis, have not chosen to do so." Data-processing, to Sammet, "has a significant intellectual component" which went unrecognized as a consequence of academia's "snob reaction" to it.[34] In particular, she cited problems of machine independence and file organization as truly challenging. There was also a more normative reason for academic criticism, she said. "COBOL was not considered very elegant . . . it was just useful. And usefulness and elegance are not necessarily the same thing."

To understand this academic reaction, I propose we treat COBOL as a technological *standard*. In an obvious sense, this was the intent behind COBOL's development—when the DoD said they would not purchase any machines not running COBOL, a *de facto* standard had been laid down for manufacturers to adhere to. But standards are also power, in two ways. As sociologists Timmermans and Epstein note, "standardization is a powerful, sometimes subtle, and sometimes

[32] Shneiderman, 350.
[33] Pratt, 405.
[34] All quotes here are from Sammet in conversation with Marcotty. For a full transcript, see Wexelblat, 266.

not-so-subtle means of organizing modern life."[35] Firstly, then, standardization is powerful because it organizes life—it provides a definition of how things *should* be. Secondly, the ability to set a standard is itself a manifestation of power, since standards usually come 'from above'. Reading COBOL as a *standard*—an attempt to reify an ideal of what a programming language *should* be—allows us to analyze its creation and reception through a new lens of power and control.

What are the results of such an approach? Seen this way, the academic hostility to COBOL is more than just a symptom of the broader business-academy drift. It is also a reflection of power—the power to determine aesthetic and functional ideals in programming languages—shifting away from the incumbent that was academia. COBOL, unlike ALGOL and LISP, was unique in its distinctly un-academic creation. Its verbosity and English-looking design were intentional choices by the Short-Range Committee that came under heavy attack from academics, one of whom sneered at "the folly of an English-looking language."[36] This is a normative critique, not a technical one. As seen above, COBOL's domain of data processing was also a focal point of criticism—to some academics, this was not what programming languages were *meant* to do. If COBOL was an attempt at standard-setting by 'business', the academic hostility to COBOL is better understood as an attempt to wrest back control of what languages should be in a changing world.

The academic hostility to COBOL can thus be understood as stemming from two key sources. First, there was a general disdain among academics towards the plebeian, 'simple' data-processing domain of COBOL. Secondly, the academic reaction itself can be read as a battle being waged over *standards* of what a programming language should be, aesthetically and functionally. The academic reaction was not in a vacuum: it was shaped by the diverging nature of the academy-business connection after the Second World War.

[35] Timmermans and Epstein, 70.
[36] Shneiderman, 351.

CONCLUSION

I began this paper by outlining COBOL's behind-the-scenes existence in the world of today, but many of the issues that swirled around it are still front and center. The gap between data-processing and academia is still relevant—although most universities now offer database modelling and information management courses, much of data-processing development still comes from outside the academy. In attempting to draw lessons from the COBOL experience, however, it is important to note that today's computing landscape is radically different from that of the 60s and 70s. The rise of the Internet, PCs, the software industry and predictive analytics have all been paradigm-shifting events that leave the COBOL heyday nearly unrecognizable, and leave the traditional academy-business-defense connections in need of reevaluation.

But that is not to say that nothing can be learnt from understanding COBOL's reception. Computing history may not repeat, but it certainly rhymes.[37] JavaScript, by some accounts, is the most popular programming language in the world today. Much like COBOL, it has borne its fair share of criticism, and from influential figures: internet pioneer Robert Cailliau once referred to JavaScript as the "most horrible kluge in the history of computing."[38] Famously replete with technical problems[39], JavaScript gives the programmer (and the academic) much to grumble about. But a great deal of the criticism JavaScript receives is along more aesthetic, philosophical and functional lines—concerns about what a web-scripting language *should* be. In writing the history of, and shaping JavaScript—as HoPL IV, scheduled for 2021, attempts to do—there is a lot to be learned from COBOL. Zooming out a little, there is also much to be gained from the programming-language-as-a-standard historiographical approach. Arguably every programming language is an ostensible standard, and many contemporary ones would benefit from close social histories that treat them as such.

None of these contemporary parallels, however, should take away from the lively, ongoing and unique history of COBOL itself. Designed as a readable, stop-gap language to be eventually discarded, COBOL exceeded all possible expectations, becoming the first language to achieve true

[37] Paraphrased from Levitsky and Ziblatt's *How Democracies Die* (2018).
[38] Wirfs-Brock, Allen, and Brendan Eich, 77:30.
[39] See wtfjs.com for a non-exhaustive list of JavaScript inconsistencies.

portability across business machines. Its development and the world it was born into was indelibly shaped by the post-war dynamics of its various communities of computing—in historicizing COBOL, we also better understand Burroughs, American Airlines, and the Department of Defense. In contrast to the enormity of its relevance, the histories written on COBOL (and programming languages in general) are few and far between. Dijkstra may not have seen anything worth learning in COBOL, but its unique history and rich legacy beg to differ.

Neel Shah
HIST-325 Final Paper

BIBLIOGRAPHY

Allen, Ben. "Common language: Business programming languages and the legibility of programming." IEEE Annals of the History of Computing 40, no. 2 (2018): 17-31.

Campbell-Kelly, Martin. "Development and structure of the international software industry, 1950-1990." Business and economic history (1995): 73-110.

Conner, Richard L. (14 May 1984). "Cobol, your age is showing". Computerworld. 18 (20): ID/7–ID/18. ISSN 0010-4841.

Dijkstra, Edsger W. "How do we tell truths that might hurt?." ACM Sigplan Notices 17, no. 5 (1982): 13-15.

Finley, Klint. "Can't File for Unemployment? Don't Blame Cobol." Wired. Conde Nast, April 22, 2020. https://www.wired.com/story/cant-file-unemployment-dont-blame-cobol/.

Hicks, Mar. "Built to last." Logic Magazine, no. 11 (2020)

James W. Cortada, Information Technology as Business History: Issues in the History and Management of Computers (Greenwood Press, 1996)

James W. Cortada, The Digital Hand: How Computers Changed the Work of American Manufacturing, Transportation, and Retail Industries (Oxford University Press, 2004)

Kurtz, Thomas E., and Richard L. Wexelblat. "History of Programming languages." (1981): 516.

MacLennan, Bruce J. 1983. Principles of Programming Languages: Evaluation and Implementation. New York, Holt, Rinehart and Winston.

Mahoney, Michael S. "What makes the history of software hard." IEEE Annals of the History of Computing 30, no. 3 (2008): 8-18.

Nofre, David, Mark Priestley, and Gerard Alberts. "When technology became language: The origins of the linguistic conception of computer programming, 1950–1960." Technology and culture (2014): 40-75.

Phillips, Charles A. (1959b). Report from Committee on Data Systems Languages. (Oral presentation to Association for Computing Machinery, Boston, Massachusetts, September 1, 1959.)

Pratt, Terrence W. 1984. Programming Languages: Design and Implementation. Second Edition, Englewood Cliffs, N.J., Prentice-Hall.

Raymond, Eric S., and Guy L. Steele, eds. The new hacker's dictionary. Mit Press, 1996.

Neel Shah
HIST-325 Final Paper

Ritasdatter, Linda Hilfling. "Unwrapping Cobol: Lessons in Crisis Computing." PhD diss., Malmö universitet, 2020.

Sammet, Jean E. "The early history of COBOL." In History of Programming Languages, pp. 199-243. 1978.

Shneiderman, Ben. "The relationship between cobol and computer science." Annals of the History of Computing 7, no. 4 (1985): 348-352.

Thomas Haigh, "Inventing Information Systems: The Systems Men and the Computer, 1950-1968", The Business History Review 75, 1(2001), 15-61.

Thomas Haigh, "The Chromium-Plated Tabulator: Institutionalizing an Electronic Revolution, 1954-1958", IEEE Annals of the History of Computing 23,4(2001), 75-104.

Timmermans, Stefan, and Steven Epstein. "A world of standards but not a standard world: Toward a sociology of standards and standardization." Annual review of Sociology 36 (2010): 69-89.

Wirfs-Brock, Allen, and Brendan Eich. "JavaScript: the first 20 years." Proceedings of the ACM on Programming Languages 4, no. HOPL (2020): 1-189.

"Academia Needs More Support to Tackle the IT Skills Gap." Academia needs more support to tackle the IT skills gap | Micro Focus. Accessed March 19, 2021. https://www.microfocus.com/about/press-room/article/2013/academia-needs-more-support-to-tackle-the-it-skills-gap/.

"As Unemployment Claims Spike, New Jersey Seeks COBOL Coders." Government Technology State & Local Articles - e.Republic. Accessed March 19, 2021. https://www.govtech.com/computing/As-Unemployment-Claims-Spike-New-Jersey-Seeks-COBOL-Coders.html.

"Computer History Museum." COBOL Tombstone. Accessed March 19, 2021. https://www.computerhistory.org/revolution/early-computer-companies/5/117/2401.

"ENIAC at Penn Engineering." Penn Engineering. Accessed March 19, 2021. https://www.seas.upenn.edu/about/history-heritage/eniac/.