Hopper and Dijkstra: Crisis, Revolution, and the Future of Programming

Sandy Payette
Cornell University

In the late 1960s, tensions were erupting in corporate and academic computing cultures in the United States and abroad with competing views about the state of computer programming and possible future implications. This article examines the computer programming cultures during this period by viewing them through the lens of two dissimilar leaders, Grace Hopper and Edsger Dijkstra, who emphasized pragmatic versus theoretical stances, respectively.

Approximately 20 years after the first programs were run on the first electronic computers, a discourse about a "software crisis" was ignited when NATO hosted a conference on the topic of software engineering that engaged university professors, computer manufacturers, and computer programmers.[1] For some, there was concern about the state of programming itself—its methods, its complexity, and its lack of theoretical underpinnings. For others, there was the specter of a labor shortage as organizations faced challenges such as increasing the supply of skilled programmers and decreasing costs during a period of unprecedented growth in commercial computing applications.

The rhetoric of crisis, revolution, and promise in the discourses of computer programming can be viewed through the lens of two highly influential leaders who were prominent during the proximate timeframe of the 1968 NATO conference. One of these leaders was female and did not attend the event; the other was male and played a significant role in the NATO conference deliberations. Grace Murray Hopper (1906–1992) and Edsger Wybe Dijkstra (1930–2002) articulated views about the role of computer programming in the domains of commercial and academic computing, respectively, shedding light on debates that were emerging about whether there was a crisis in computer programming in the 1960s and what should be done about it.

I have selected Hopper and Dijkstra as representatives of different discourse communities[2] that offered alternative views on the "right" direction for the future of computer programming. As Turing Award winner Robert Floyd observed, "In computer science, one sees several such communities, each speaking its own language and using its own paradigms."[3] While Floyd invoked Kuhn's theory of paradigm,[4] Kuhn speaks of paradigm in the context of scientific revolution, where a new theory overtakes the old. In this process, concepts and terms become "incommensurable," leading to some terms dropping out of use and others being redefined to be compatible with the new scientific theory. However, Kuhn's position on the language of paradigm shift is limited in its ability to serve as a conceptual tool for understanding multifaceted discourses of computer programming communities.

Accordingly, I am influenced by Paul Edwards' expansive definition of discourse as "a background of assumptions and agreements about how reality is to be interpreted and expressed, supported by paradigmatic metaphors, techniques and technologies and potentially embodied in social institutions."[5] In *The Closed World,* Edwards argued that to fully understand the computer, we must uncover the social and political meanings that include the conceptual models we build, the metaphors we use, and the experiences we have with the computer. While Kuhn's paradigm is useful, Edwards argues that it will not accommodate the complexity and multiplicity of discourses that coexist in different communities; such discourses sometimes compete and other times intersect and align, depending on the context.[6] Edwards is influenced by Foucault in his accommodation of this complexity.[7]

 Published by the IEEE Computer Society 1058-6180/14/$31.00

For my purposes, discourses can illuminate ongoing sites of struggle by revealing power dynamics and claims of legitimacy in different communities. Discourses can reveal values and beliefs and uncover the origins of tensions that endure over time. In this article, I use discourses of computer programming to help reveal tensions between practice and theory during a critical period in the history of commercial and academic computing. Specifically, I use Hopper and Dijkstra as models and exemplars that align with the dominant values and beliefs of their primary communities—industry and academia, respectively. I focus on representations of programming that brought both meaning and resonance to an evolving communicative system. In this way, my approach is also similar to the "symbolic heterogeneous engineering" method of Charles Bazerman, who unearthed tensions between entrepreneurial innovators and scientists in the discourses surrounding Edison and the history of electric light.[8]

When examining the artifacts of computer programming discourse, it is important to acknowledge the different timeframes and institutional contexts that Hopper and Dijkstra inhabited before we consider the overlapping decades of their words and practices. After a brief overview of their early histories and highlights of their lifelong careers, I will focus primarily on the period from the 1950s to the early 1970s, which encompasses the time when they produced their major works. I will also examine statements Hopper and Dijkstra made in the 1970s and 1980, referring to both early computing and this focal period. My sources include public speeches, published documents, meeting minutes, and occasionally, their personal reflections published in later writings and oral histories. During the focal period, they were both persuasive leaders for their primary constituencies—for Hopper, practitioners in commercial and military institutions, and for Dijkstra, theoreticians of computer science in academic departments. The broader historical context of Hopper and Dijkstra also highlights the cultural complexities of gender in computer programming, a durable phenomenon that continues in contemporary computer programming cultures.

Historical Context: Hopper and Dijkstra

An examination of Hopper and Dijkstra's histories reveals that these two pioneers of programming were from different generations and had radically different entry points into computing, which informs our understanding of the origins of the attitudes and beliefs that reverberate through their long careers.

Hopper's entry into computing was an unexpected consequence of fulfilling her personal goal to serve her country during World War II. After receiving a PhD in mathematics from Yale University and embarking on a first career as professor at Vassar College, she decided to leave academia to join the United States military. In 1944, Hopper began her computing career at Harvard's Computation Laboratory, serving as a military officer under Howard Aiken,[9] where she became one of the first programmers on the Automatic Sequence Controlled Calculator, known as the Mark I computer.[10] Hopper was a major innovative force in programming the new Harvard computer before the term "programming" had entered the computing lexicon.[11]

After her work at Harvard, Hopper joined the Eckert-Mauchly Computer Corporation (EMCC) in 1949, choosing this small company over many other opportunities because of its exciting startup environment[12] in the development of the Univac, the first commercially viable electronic computer in the United States. In the 1950s, while at EMCC and later Remington Rand, Hopper developed many foundational building blocks for human-oriented computer languages (such as subroutines and pseudocode) that led to her invention of the A-0, which is recognized as the first compiler for a computer.[13] Hopper's impact continued for decades in the area of programming languages, with her notable development of FLOW-MATIC, which served as the dominant model for Cobol, the language for which Hopper is most commonly associated. Hopper served in the US Naval Reserve throughout her life and career, initially retiring from the military at age 60, but then being called back to active duty for her expertise in computer programming. She ultimately retired as a rear admiral in 1986, the oldest officer on active duty. She was the recipient of many awards in her lifetime, including the paradoxically entitled computer science "Man of the Year" Award from the Data Processing Management Association (DPMA),[14] the National Medal of Technology, and recognition as Distinguished Fellow of the British Computer Society.[15]

Dijkstra's entry into computing began approximately 10 years after Hopper's, with a love for computer programming and a determination to make computer science "an

intellectually respectable discipline."[16] Dijkstra was pursuing his PhD in theoretical physics while also working as programmer at the Mathematical Center in Amsterdam. He was displeased with the precedent established by the military and commercial actors who were responsible for the design of the first generations of electronic computers, which in turn influenced the approaches that could be taken by programmers. In 1972, he reflected on the release of the third generation of such computers in the 1960s, asserting that

> It was only reasonable to expect that such machines would flood the computing community, and it was therefore all the more important that their design should be as sound as possible. But the design embodied such serious flaws that I felt that with a single stroke the progress of computing science had been retarded by at least ten years.[17]

Dijkstra discovered the intellectual possibilities inherent in computer programming and was determined to correct the mistakes he believed were made in the past. He developed new approaches to structured programming that were grounded in theory, formal reasoning, and mathematical provability of program correctness.[18] He is also known for key/foundational works in multiprogramming,[19] self-stabilizing systems in a distributed environment,[20] and the shortest-path problem for a graph.[21] Winner of the prestigious Turing Award in computer science, Dijkstra reflected in his 1972 Turing lecture that a turning point in his life was when his mentor asked him whether he might become "one of the persons called to make programming a respectable discipline in the years to come."[22] Dijkstra took up the challenge and is recognized for his intellectual impact and leadership in the emergence of computer science as an academic discipline.

Urgency and Innovation versus Crisis and Revolution

Like other programmers who appeared in the 1940s, such as the women who worked on the ENIAC,[23] Hopper was driven by demands for wartime calculation that required tenacity and innovation to overcome formidable obstacles in making the new machines perform. A discourse of urgency and continuous innovation had its roots in this early stage of computer programming that was characterized by cleverness, trial and error, and quick response. In the 1950s and 1960s, the discourses of innovation had a new sense of urgency focused on the priorities of a budding commercial computing industry. In the context of Univac, the new topics included the development of the first compilers and human-oriented programming languages. Regarding the environment of the period, Hopper reflected in a 1978 keynote address, "Those were precarious days. We used to say that if UNIVAC I didn't work, we were going to throw it out one side of the factory, which was a junk yard, and we were going to jump out the other side, which was a cemetery!"[24]

In academic circles, another discourse began to bubble up, led by those who were in a position to critique and respond to the first two decades of military and commercial programming. This discourse was an expression of new research that was underway that could demonstrate radical new approaches to programming based on science and formal methods. When in the late 1960s the term "software crisis" emerged, we can discern competing views about the appropriate future direction for programming. While Hopper and Dijkstra both had their eyes set on improving the quality of computer programs, an examination of their discourses helps reveal a tension. Hopper and Dijkstra serve as proxies for the two sides of a debate about whether there was a crisis in programming and, if so, what should be done about it.

Pragmatic "Can-Do" Rhetoric

If we examine the history of Hopper's work in computer programming that began with Mark I/II, and especially the body of work done on Univac in the 1950s and 1960s, we see consistent themes of pragmatism, urgency, opportunity, and collaboration. In the 1970s, Hopper's statements about these years express an action orientation with a "can-do" rhetoric and a sense of getting things done. For example, in 1972, Hopper acknowledged the challenges presented by the wartime context of early programming:

> Don't forget that Mark II was built under wartime pressure. Get that thing built fast. Use existing components. Nothing new. Get it built. Get it running. It was critical…. But there was no theorizing, there was no higher mathematics. There was no future of computers, there was nothing but get those problems going, and what the computer was doing.[25]

We can also see similar elements of the "can-do" and "let's get going" rhetoric in her

projections about the future of computing. For example, in a 1974 conference of the Data Processing Institute, Hopper spoke about possible futures and asserted,

> If man could build Stonehenge 4,000 years ago, and, it is part of the heritage of every one of us, we can do it today. But, it's going to take hard work, it's going to take a lot of learning, it's going to take standards, it's going to take modules and all the new things that you have been hearing about. So, we better get with it now.[26]

In her later reflections, Hopper offered a pragmatic account of how she developed the first compiler for Univac in the 1950s. In 1980, Hopper claimed that she was always inclined to take a practical approach to innovation and was driven by common sense to solve a problem:

> I think I always looked for the easiest and best, most accurate way of getting something done. I don't use high-level theory. I use very basic common sense—the whole original concept of the compiler is the most common sense thing I ever heard of.... It was so common sense that it had to be done.[27]

Crisis and Revolution Rhetoric

In his later years, Dijkstra reflected on computer programming of the 1960s, noting,

> In retrospect it looks like a kaleidoscopic activity! It was. The turning point had been the NATO conference on Software Engineering.... It was there and then that the so-called "Software Crisis" was admitted and the condition was created under which programming as such could become a topic of academic interest.[28]

The organizers of the 1968 NATO conference chose the term *software engineering* to be "deliberately provocative," to signal that programming was lacking the scientific and formalized processes found in other areas of science and engineering.[29] While the question of whether there was, indeed, a crisis was debated at the time, the reframing of programming as a type of engineering was an important development.[30] Scholars have agreed that the impact of the NATO conference was substantial because it provided the foundation for the ongoing discourse of a crisis in programming, along with the proposed remedy that software development be based on principles of engineering, both of which have been enduring.[31] In the 1970s, Dijkstra was a notable contributor to the discourse of crisis as well as to the dissemination of the term *software engineering*.[32]

Dijkstra often spoke with a sense of regret about missteps of the past in computer programming, arguing for the need to radically change course. In 1970, he sought an academic ideal and fully embraced the intellectual challenges inherent in programming.

> We should recognize that already now programming is much more an intellectual challenge: the art of programming is the art of organizing complexity, of mastering multitude and avoiding its bastard chaos as effectively as possible.[33]

Dijkstra was particularly concerned with delineating the boundaries of computer science as a new discipline by moving programming in a scientific direction grounded in abstraction, formal logic, and provability. In his reflections, he expressed regret about the formative period of programming, likening early conceptions of the field of "computing science" as analogous to referring to surgery as "knife science." He argued that this early legacy influenced people to associate computer science more with computing machines than with the abstraction of programming.[34] In 1972, we can see murmurings of crisis rhetoric in statements Dijkstra made in his Turing lecture:

> In a sense the electronic industry has not solved a single problem, it has only created them—it has created the problem of using its products.... and it was the poor programmer who found his job in this exploded field of tension between ends and means.[17]

Later, with regard to the first generation of programming languages, Dijkstra's reflections were dramatic and sometimes combative. He referred to Fortran as the "infantile disorder'" and PL/I as "the fatal disease." Consistent with these metaphors of disability, he said "the use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence."[35] In contrast, he described Algol as a miracle, and he was especially impressed with its formal grammar and its introduction of recursion into programming.[36]

Bolstered by a discourse of software crisis and anticipating a new revolutionary era of logic and reasoning, Dijkstra said in his 1972 Turing lecture, "Such a drastic change in such

a short period of time would be a revolution…. But we all know that sometimes revolutions do take place!"[37]

Debugging versus Proving

The tensions represented by Hopper's and Dijkstra's views on programming are revealed more specifically in an analysis of the debugging versus proving debate. In the 1960s, the process of debugging had come to the forefront of discussions about improving program quality and was an issue of concern both at the NATO conference and in industry. Hopper and Dijkstra were representative of two sides of the debate, but they agreed that it was critical for programmers to create well-designed programs that would solve well-defined problems. For Hopper, the process involved the use of tools such as flow charts, followed by coding using structured programming languages, and ultimately compiling programs into machine code that could then be run, tested, and debugged.[38] Hopper viewed debugging as a necessary part of the program development process; however, she also advocated for new methods that could make the process easier for programmers. She envisioned debugging capabilities that could be built into "precompilers." She argued that in addition to making the programmer's job easier by removing bugs before compilation, a precompiler could impose coding standards that would result in higher-quality programs.[39]

In contrast to Hopper's approach, Dijkstra made a strident distinction between "pragmatic programming" and scientifically designed programs, arguing that existing practice should be replaced with a new process based on formal logic and mathematical proofs. Dijkstra thought the new scientific approach should not resemble "iterative design"—a cycle of design-program-test-debug—that he viewed as the paradigm for the pragmatist. Instead, he argued that "[i]f you want more effective programmers, you will discover that they should not waste their time debugging—they should not introduce bugs to start with."[37]

Dijkstra's radical and exciting proposition was that "provably correct" programs would not have to be debugged because, in fact, they would not even have to be run to know that they were error free![40] This approach was highly appealing to a theory-oriented constituency that could now explore programming from a more abstract and mathematical perspective. In contrast, in a 1978 ACM keynote, Hopper critiqued academic computer science, noting that a turn in programming that would exclusively value higher abstraction would be disadvantageous because it would require all programmers to write code in the model of the mathematical logician. Reflecting on her motivations, Hopper said

> We were also endeavoring to provide a means that people could use, not that programmers or programming language designers could use. But rather, plain, ordinary people, who had problems they wanted to solve….[41]

At the same time, the theoretical achievements of Dijkstra and other theory-oriented computer scientists resulted in significant advances in programming languages, especially relevant to complex software systems that demand correctness and the highest degrees of reliability.

Reflections on Human Factors

Hopper and Dijkstra engaged in discourses regarding the relative importance of scientific and human factors in computer programming. At the time of the NATO conference, Hopper advocated that

> We have to keep in mind in any given situation the total environment, which involves not just the programming language, but involves documentation, and it involves the people that are using it and the managers of those people.[42]

In contrast, Dijkstra criticized proposals that advocated for the introduction of management and communication components into software engineering education curricula. In a 1982 letter to H.D. Mills, Dijkstra dramatically argued that the recent "fashion" for emphasizing the "soft sciences" would present a "threat to our civilization":

> The most blatant example of superficiality is probably their argument in favour of communication skills, as they refer to "the software engineer's need to communicate with a wide range of people and machines". As a piece of hilarious nonsense I think that this is only surpassed by "The education of a computer."[43]

In this letter, we see Dijkstra making reference to Hopper's 1952 article about the first compiler, entitled "The Education of a Computer." It is notable that in 1999, more than 40 years after Hopper published this paper,

Dijkstra reasserted his criticism of both Cobol and Hopper's paper.

> It was the time of rampant anthropomorphism that would lead to the false hope of solving the programming problem by the verbosity of COBOL and would seduce Grace M. Hopper to write a paper titled "The Education of a Computer." Regrettably and amazingly, the habit lingers on….[44]

Despite Hopper's paper being historic and providing a rhetorical analogy of her compiler as providing the computer with the equivalent of a college education, Dijkstra continued to find it offensive. He considered anthropomorphic metaphors to be anathema to scientific approaches, declaring in 1975 that they are "a symptom of professional immaturity."[45] But more substantively, Dijkstra appeared to frame Hopper's work as paradigmatic of an unscientific approach to programming.

Although both Hopper's and Dijkstra's statements were often notably dramatic, their themes were consistent over decades, resonating with different discourse communities. Hopper's discourse tended to resonate within institutional contexts that demanded communication and negotiation with diverse professionals. Dijkstra's was particularly resonant in academic research contexts that valued theoretical rigor and an orientation toward formalism and abstraction. Two of their later statements from the 1980s reflect these ideals:

> Hopper (1980): I kept calling for more user friendly languages. I've always tried to do that, that's why I want these other languages that are aimed at people. Most of the stuff we get from academicians, computer science people, is in no way adapted to people.[46]

> Dijkstra (1986): I do see the possibility of a fascinating future…. It is a future in which programs will display all the beauty of a crisp argument, and in which the dictionaries will no longer define mathematics as the "abstract science of space, number, and quantity"… but as the "art and science of effective reasoning."[47]

Unfortunate Absences and Paradoxes

Discourses can help us reflect on the origins of social and political concerns surrounding computing and help reveal paradoxes when ideals and actual experiences don't align. For example, considering the importance of women in early programming contexts and the absence of women at the NATO conference, we can contemplate a gender paradox in the history of computing. The

Figure 1. All male attendees at the 1968 NATO conference. No female computer experts were present at Garmisch.

1968 NATO conference proceedings indicate that no female computer experts were present at Garmisch. Although the delegation was selected to represent "a wide variety of backgrounds,"[48] the NATO proceedings still acknowledged that its primary focus was on topics deemed crucial to software engineering, and thus the conference "did not attempt to provide a balanced review of the total state of software, and tends to understress the achievements of the field."[49] The NATO conference gave meaning to the term *software engineering,* yet the prima facie profile of the conference report and photos has inscribed the discourse of an exclusively male expert cohort surrounding the this new direction for programming (see Figure 1).[50]

Particularly ironic is Hopper's absence at a conference that was sponsored by NATO, an organization concerned with military defense. Hopper's active military status in the US Navy was reinstituted in 1967, just before the NATO conference, when she was called back to active duty to serve as the director of the Navy's programming languages group. We get a glimpse of Dijkstra's attitudes and reactions to Hopper and her military background when he commented about Hopper in his 1973 trip report for the IUCC Computer Science Colloquium, where they were both invited speakers. Dijkstra recalled that

> Captain Hopper spoke officially about "Programming Languages"; here [*sic*] real subject was how she had acted as midwife to COBOL and she talked more about the Pentagon and the U.S. Navy than about programming.[51]

We see evidence of Dijkstra's impatience with Hopper's institutional context and military-infused discourse, as well as an element of gendered discourse in his characterization of

Hopper as a "midwife," a term that suggests a feminine support role rather than leadership.

Conclusion

Regardless of whether we are more oriented toward communities whose discourses are of the pragmatic orientation, as in the Hopper model, or those of a theoretical orientation such as in the Dijkstra model, it is worthwhile to consider the role of historic discourses in illuminating social issues and untangling the meanings, values, and beliefs that underlie different communities that struggle for power and legitimacy. In this article, tensions such as theory versus practice, debugging versus proving, craft versus engineering, and crisis versus revolution were explored through analysis of discourse. Also, consideration of the artifacts of the 1968 NATO conference revealed that a discourse on gender and computing may not be directly spoken, yet silence can raise important questions for scholars to contemplate when they observe who and what were missing from the record of historic communicative events. The 1950s and 1960s were a time of notable participation of women in computing,[52] and a few notable women gained recognition, such as Hopper who Mahoney described as having "achieved something akin to canonization in her own lifetime."[53] Nevertheless, women still operated in a culture with a dominant masculine bias.[54]

After the NATO conference, the discourse of software engineering illuminated beliefs that computer programming should move toward formality and predictability, following the models of science and engineering.[55] At the same time, however, the discursive turn from computer programming to software engineering associated programming with a bastion of masculinity and machines[56] that was dominated by male voice and imbued with silences about gender. The discourse that associated software with engineering[57] permitted the social construction of gender found in engineering to seep into computer programming culture. Like other histories of professionalization,[58] computing workplaces began to establish boundaries between low- and high-status roles, with proportionally more women represented in lower-status positions.[59]

Looking ahead, an appreciation of the discourses of the computer programming can provide a foundation for grappling with both the historic and contemporary social and political issues that are embedded in decades of programming history and that continue to have meaning and resonance in emerging discourses of the Internet, cyberinfrastructure,[60] and knowledge infrastructures[61] both today and in the future.

I will close with a notable contemporary discourse of programming—free and open source software—where we see an extreme gender disparity reappearing in another "revolutionary" context. Open source is a community-driven model of software development that produces software that is distributed free of charge and code that is modifiable by anyone who wishes to participate. Open source software underlies many essential components of the Internet and the Web, infrastructures that increasingly affect many dimensions of our lives. The ethos that motivates open source is freedom and democracy[62] (for example, "free as in free speech, not as in free beer"[63]) and the power of peer production[64] (such as "a collective barn raising"[65]), a paradox when such powerful metaphors can leave the movement's lack of gender diversity hiding in plain sight. This pro-social discourse eclipses the fact that the percentage of female open source developers has been estimated at 1.5 percent,[67] a far cry from the already low percentage of female open source developers in commercial and academic computing workplaces.[68] This is just one of many examples that present interesting opportunities for future research that examines the power and the paradoxes of discourses in computing.

References and Notes

1. First, see the proceedings of the 1968 NATO conference found at http://homepages.cs.ncl.ac.uk/brian.randell/NATO/: P. Naur and B. Randell, *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee*, 7–11 Oct. 1968. Then, for scholarly analysis of the meaning and impact of the conference, see N. Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise*, MIT Press, 2010, chap. 8; also see J. Abbate, *Recoding Gender: Women's Changing Participation in Computing*, MIT Press, 2012, pp. 73–111; M.S. Mahoney and T. Haigh, *Histories of Computing*, Harvard Univ. Press, 2011, pp. 34, 52, 84, 92; M. Campbell-Kelly and W. Aspray, *Computer: A History of the Information Machine*, Basic Books, 1996; E.N. Yourdon, *Classics in Software Engineering*, Yourdon Press, 1979; N. Wirth, "A Brief History of Software Engineering," *IEEE Annals of the History of Computing*, vol. 30, no. 3, 2008, pp. 32–39. A frequently cited paper that was presented in 1968 at the NATO

conference is M.D. McIlroy, "Mass Produced Software Components," *Software Engineering: Report on Conference Sponsored by NATO Science Committee*, 1969, pp. 138–155.

2. On discourse communities, see R.R. Kline, "Cybernetics, Management Science, and Technology Policy: The Emergence of 'Information Technology' as a Keyword, 1948–1985," *Technology and Culture*, vol. 47, no. 3, 2006, p. 514. Also see the introduction in R. Oldenziel, *Making Technology Masculine: Men, Women, and Modern Machines in America, 1870–1945*, Amsterdam Univ. Press, 2004. For general definitions, see P. Baker and S. Ellece, *Key Terms in Discourse Analysis*, Continuum International Publishing Group, 2011, p. 33.
3. R.W. Floyd, "The Paradigms of Programming," *Comm. ACM*, vol. 22, no, 8, 1979, p. 456.
4. T. Kuhn, *The Structure of Scientific Revolutions*, Univ. of Chicago Press, 1970. To understand the evolution of Kuhn's theory of incommensurability, see T. Kuhn, *The Road Since Structure: Philosophical Essays, 1970–1993, with an Autobiographical Interview*, J. Conant and J. Haugeland, eds., Univ. of Chicago Press, 2000. When considering discourses in the history of programming, Kuhn's notion of upheaval and how old terms become redefined to fit a new paradigm imposes constraints for considering discourses that evolve, intersect, and coexist over time. I look at discourses as reflections of enduring power struggles and tensions. I am more influenced by Foucault's linking of knowledge and power, and the notion of evolving, coexisting discourses. See M. Foucault, *The Archaeology of Knowledge*, Tavistock Publications, 1972.
5. P.N. Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America*, MIT Press, 1996, p. 34.
6. See Edwards, *The Closed World*, pp. 30–41, for a full summary of his approach to discourse, especially the limitations of Kuhn and the influences of Wittgenstein and Foucault.
7. Note that Foucault said, "Discourses are tactical elements or blocks operating in the field of force relations; there can exist different and even contradictory discourses with the same strategy; they can, on the contrary, circulate without changing their form from one strategy to another, opposing strategy." See M. Foucault, *The History of Sexuality: Volume* 1, Vintage Books, 1978/1990, pp. 101–102.
8. C. Bazerman, *The Language of Edison's Light*, MIT Press, 1999. Methodologically, Bazerman placed his lens directly upon a system of "texts" that reveal communicative events and symbols that gave "presence, meaning, and value to a technological object or process." His sources included lab notebooks, newspaper stories, scientific publications, diagrams, and patent applications. He argued that Edison's use of language around electric light was as important as the demonstrations of the technology itself. These discourses reveal a tension between the roles of inventor versus scientist, where Edison is seen as the inventor and persuasive entrepreneur who, despite the impact of his work, had to strategically position to gain legitimacy amid a community of academics and professional societies who valued pure science (pp. 139–140).
9. For a history of Aiken and Mark I, see I.B. Cohen, *Howard Aiken: Portrait of a Computer Pioneer*, MIT Press, 1999.
10. H. Aiken and G. Hopper, "The Automatic Sequence Controlled Calculator I," *Electrical Engineering*, vol. 65, no. 8–9, 1946, pp. 384–391. Also see parts II/III in vol. 65, no. 10, pp. 449–454, and vol. 65, no. 11, pp. 522–528.
11. Regarding Hopper's role, see K. Beyer, *Grace Hopper and the Invention of the Information Age*, MIT Press, 2009; K.B. Williams, *Grace Hopper: Admiral of the Cyber Sea*, Naval Inst. Press, 2004.
12. Beyer, *Grace Hopper and the Invention of the Information Age*, p. 172.
13. See key works by Hopper in the 1950s, including G.M. Hopper, "The Education of a Computer," *Proc. ACM Nat'l Meeting*, 1952, pp. 243–249; G.M. Hopper, "Compiling Routines," *Computers and Automation*, vol. 2, no. 4, 1953, pp. 1–5; G.M. Hopper, "Automatic Coding for Digital Computers," *Computers and Automation*, vol. 4, no. 9, 1955, p. 21; G.M. Hopper, "From Programmer to Computer," *Industrial & Eng. Chemistry*, vol. 50, no. 11, Nov. 1958, p. 1661.
14. In 1969, Hopper received the first "Man of the Year" award from the Data Processing Management Association, currently known as Association of Information Technology Professionals, www.aitp.org/?page=aitphistory.
15. See the roster of distinguished fellows of the British Computer Society at www.bcs.org/content/conwebdoc/1650. It is notable that Hopper was the third fellow in 1973 and Dijkstra was the first fellow in 1971.
16. E.W. Dijkstra, "The Humble Programmer," *Comm. ACM*, vol. 15, no. 10, 1972, pp. 859–860.
17. Dijkstra, "The Humble Programmer," p. 861.
18. See E.W. Dijkstra, "Structured Programming," *Software Engineering Techniques: Report of NATO Science Committee*, 1970; E.W. Dijkstra, "Notes on Structured Programming," *Structured Programming*, O.J. Dahl, E.W. Dijkstra, and C. Hoare, eds., Academic Press, 1972.
19. E.W. Dijkstra, "The Structure of the 'THE' – Multiprogramming System," *Comm. ACM*, vol. 11, no. 5, 1968, pp. 341–346.

20. E.W. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control an On-Site Data Management System Application," *Comm. ACM*, 17, no. 11, 1974, pp. 643–644.
21. E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, vol. 1, 1959, pp. 269–271.
22. Dijkstra, "The Humble Programmer," p. 860.
23. Operation of the ENIAC was initially modeled upon assumptions based on an earlier era of "human computers," who were predominantly female and who manually performed mathematical calculations. See Abbate, *Recoding Gender*, pp. 11–38; D.A. Grier, *When Computers Were Human*, Princeton Univ. Press, 2005; J.S. Light, "When Computers Were Women," *Technology and Culture*, vol. 40, no. 1, 1999, pp. 455–483. See also P. Ceruzzi, "When Computers Were Human," *IEEE Annals*, vol. 13, no. 3, 1991, pp. 237–244; W.B. Fritz, "The Women of ENIAC," *IEEE Annals*, vol. 18, no. 3, 1996, pp. 13–28.
24. G.M. Hopper, "Keynote Address: 1978 ACM SIGPLAN History of Programming," *History of Programming Languages*, R.L. Wexelblat, ed., ACM, 1978, p. 8.
25. B. Luebbert and H. Tropp, "Grace Murray Hopper Interview July 5, 1972," Computer Oral History Collection, 1969–1973, 1977, Smithsonian Nat'l Museum of Am. History, 1972, pp. 4, 10.
26. G.M. Hopper, "Technology: Future Directions," *Proc. Conf. Data Processing Inst.*, 1974, p. 14.
27. A. Pantages, "Oral History of Captain Grace Hopper," Computer History Museum, 1980, p. 20, http://archive.computerhistory.org/resources/text/Oral_History/Hopper_Grace/.
28. E.W. Dijkstra, "Computing Science: Achievements and Challenges," *ACM SIGAPP Applied Computing Rev.*, vol. 7, no. 2, 1999, p. 8. Also see transcript of handwritten original in E.W. Dijkstra Archive, EWD 1284, www.cs.utexas.edu/users/EWD/transcriptions/EWD12xx/EWD1284.html.
29. With regard to the term being provocative, see the introduction to 1968 NATO report; also see B. Randell, "Software Engineering in 1968," *Proc. 4th Int'l Conf. Software Eng.* (ICSE), 1979, pp. 1–10; and B. Galler, "ACM President's Letter: NATO and Software Engineering?" *Comm. ACM*, vol. 12, no. 6, 1969, p. 301.
30. See analysis of software crisis in Abbate, *Recoding Gender*, pp. 73–111.
31. See especially Ensmenger, *The Computer Boys Take Over*, chap. 8. Ensmenger observed that from 1968 to 1972, which is the time period of both NATO conferences, that "the rhetoric of the crisis became firmly entrenched in the vernacular of commercial computing" (p. 195).
32. For example, see T. Haigh, "Crisis, What Crisis? Reconsidering the Software Crisis of the 1960s and the Origins of Software Engineering," *Proc. 2nd Inventing Europe/Tensions of Europe Conf.*, June 17–20, 2010, p. 2. Also regarding Dijkstra's role, see T. Haigh, "Dijkstra's Crisis: The End of Algol and Beginning of Software Engineering 1968–72," SOFT-EU Project Meeting, 2010.
33. Dijkstra, "Notes on Structured Programming," p. 6.
34. E.W. Dijkstra, "On a Cultural Gap," *The Mathematical Intelligencer*, vol. 8, no. 1, 1986, p. 49.
35. E.W. Dijkstra, "How Do We Tell Truths That Might Hurt?" *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, 1982, p. 130; also E.W. Dijkstra, "How Do We Tell Truths That Might Hurt?" E.W. Dijkstra Archive, EWD 498, http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD498.html.
36. Summarized in Dijkstra, "Computing Science: Achievements and Challenges," pp. 4–5.
37. Dijkstra, "The Humble Programmer," p. 863.
38. Hopper, "Automatic Coding for Digital Computers," pp. 21–22. Note how Hopper articulated this process of program development and debugging.
39. "CODASYL, Conference on Data Systems Languages," Computer Oral History Collection, 1969–1973, 1977, Smithsonian Nat'l Museum of Am. History, 1969, pp. 38–39.
40. Again, see Dijkstra, "Notes on Structured Programming," chap. 1.
41. G.M. Hopper, "Keynote Address: 1978 ACM SIGPLAN History of Programming," *History of Programming Languages*, R.L. Wexelblat, ed., ACM, 1978, p. 11.
42. CODASYL, 1969, p. 54.
43. E.W. Dijkstra, "To H.D. Mills, Chairman Software Methodology Panel," *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, 1982, pp. 220–222. Also see E.W. Dijkstra Archive, EWD 575, www.cs.utexas.edu/users/EWD/transcriptions/EWD05xx/EWD575.html.
44. Dijkstra, "Computing Science: Achievements and Challenges," p. 3. Also see transcript of handwritten original in E.W. Dijkstra Archive, EWD 1284, www.cs.utexas.edu/users/EWD/transcriptions/EWD12xx/EWD1284.html.
45. Dijkstra, "How Do We Tell Truths That Might Hurt?" 1982, p. 130. For original 1975 quote, see transciption in E.W. Dijkstra Archive, EWD 498, https://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD498.html.
46. Pantages, "Oral History of Captain Grace Hopper," p. 11.
47. Dijkstra, "On a Cultural Gap," p. 52.
48. Randell, "Software Engineering in 1968," p. 6.

49. See NATO Report on Software Engineering, 1968, p. 3.
50. See NATO conference artifacts available at Brian Randell's website at http://homepages.cs.ncl.ac.uk/brian.randell/NATO/.
51. "Trip Report I.U.C.C.Colloquium. Canterbury, 18th–21st Sept. 1973," E.W. Dijkstra Archive, EWD 389, www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD389.html.
52. Especially see Abbate, *Recoding Gender*, chap. 2. Also, J. Abbate, "The Pleasure Paradox: Bridging the Gap Between Popular Images of Computing and Women's Historical Experiences," *Gender Codes: Why Women Are Leaving Computing*, T.J. Misa, ed., IEEE CS and John Wiley & Sons, 2010, pp. 213–226.
53. Mahoney and Haigh, *Histories of Computing*, p. 106.
54. For examples of this argument, see Ensmenger, *The Computer Boys Take Over*; T.J. Misa, "Gender Codes: Lessons from History," *Gender Codes*, pp. 251–263; M S. Mahoney, "Boy's Toys and Women's Work: Feminism Engages Software," *Histories of Computing*, pp. 106–117. Also see the discussion of the history of women and software in N. Ensmenger and W. Aspray, "Software as Labor Process," *Mapping the History of Computing: Software Issues*, R.U. Hashagen, A. Keil-Slawik, and E. Norberg, eds., 2003, pp. 139–165.
55. For a survey of activities around software engineering, see B.W. Boehm, "Software Engineering," *IEEE Trans. Computers*, vol. 25, no. 12, 1976, pp. 1226–1241.
56. See Oldenziel, *Making Technology Masculine*, pp. 70–78, for an early history of engineering.
57. Also for a recent history of women in engineering, see A.S. Bix, *Girls Coming to Tech!?: A History of American Engineering Education for Women*, MIT Press, 2013.
58. A classic work in the area of professionalization is A. Abbott, *The System of Professions*, Univ. of Chicago Press, 1988.
59. Regarding professionalization in computing, see N. Ensmenger, "The Question of Professionalism in the Computer Fields," *IEEE Annals*, vol. 23, no. 4, 2001, pp. 56–73.
60. See D.E. Atkins et al., *Revolutionizing Science and Engineering Through Cyberinfrastructure? Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructue*, NSF, 2003.
61. On knowledge infrastructure, see P.N. Edwards et al., *Knowledge Infrastructures: Intellectual Frameworks and Research Challenges*, NSF, 2013, http://hdl.handle.net/2027.42/97552.
62. See especially C.M. Kelty, *Two Bits: The Cultural Significance of Free Software*, Duke Univ. Press, 2008.
63. R.M. Stallman, *Free Software, Free Society: Selected Essays of Richard M. Stallman*, Free Software Foundation, 2010, p. 3.
64. Y. Benkler, *The Wealth of Networks: How Social Production Transforms Markets and Freedom*, Yale Univ. Press, 2006.
65. Y. Benkler and H. Nissenbaum, "Commons-Based Peer Production and Virtue," *J. Political Philosophy*, vol. 14, no. 4 2006, pp. 394–419.
66. The statistic on women in Floss was first reported by D. Nafus, J. Leach, and M. Strathern, "Free / Libre and Open Source Software?: Policy Support FLOSSPOLS," *Gender Issues*, vol. 16, Mar. 2006, pp. 1–75. For a recent discussion of the gender dynamics of open source, also see J. Reagle, "Free as in Sexist? Free Culture and the Gender Gap," *First Monday*, vol. 18, no. 1, 2013; http://firstmonday.org/ojs/index.php/fm/article/view/4291/3381.
67. Well-documented data show a decline, since the 1980s, of the percentage of undergraduate degrees in computer science granted to women, as well as a shrinking proportion of women in the workplace holding key roles such as programmer and systems analyst. For summary of data trends, see C. Hill, C. Corbett, and A. St. Rose, "Women and Girls in Science, Technology, Engineering, and Mathematics," *Why So Few? Women in Science, Technology, Engineering, and Mathematics*, research report, Am. Assoc. of Univ. Women (AAUW), 2010. Also, see C.C. Hayes, "Computer Science: The Incredible Shrinking Woman," *Gender Codes: Why Women Are Leaving Computing*, John Wiley & Sons, 2010, pp. 25–49. The implications of the missing perspectives of women include a lack of diversity in the development of software that increasingly affects our lives as well as economic disadvantages to women who miss out on opportunities for employment in positions that offer creativity, power, and financial benefit.

Sandy Payette is a PhD candidate in the Department of Communication at Cornell University and was the founding CEO of DuraSpace (http://duraspace.org), a nonprofit organization that provides open source software aimed at preserving the world's intellectual, cultural, and scientific heritage. Her research interests include the history of computing, information policy, and knowledge infrastructures. Payette also has an MBA in information systems from the State University of New York at Binghamton. Contact her at sdp6@cornell.edu.