CODE NATION

Personal Computing and the Learn to Program Movement in America

Michael J. Halvorson

ASSOCIATION FOR COMPUTING MACHINERY

Code Nation

ACM Books

ACM Books is a series of high-quality books for the computer science community, published by ACM in collaboration with Morgan & Claypool Publishers. ACM Books publications are widely distributed in both print and digital formats through booksellers and to libraries (and library consortia) and individual ACM members via the ACM Digital Library platform.

Computing and the National Science Foundation, 1950–2016: Building a Foundation for Modern Computing
Peter A. Freeman, *Georgia Institute of Technology*
W. Richards Adrion, *University of Massachusetts Amherst*
William Aspray, *University of Colorado Boulder*
2019

Providing Sound Foundations for Cryptography: On the work of Shafi Goldwasser and Silvio Micali
Oded Goldreich, *Weizmann Institute of Science*
2019

Concurrency: The Works of Leslie Lamport
Dahlia Malkhi, *VMware Research* and *Calibra*
2019

The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!
Ivar Jacobson, *Ivar Jacobson International*
Harold "Bud" Lawson, *Lawson Konsult AB (deceased)*
Pan-Wei Ng, *DBS Singapore*
Paul E. McMahon, *PEM Systems*
Michael Goedicke, *Universität Duisburg–Essen*
2019

Data Cleaning
Ihab F. Ilyas, *University of Waterloo*
Xu Chu, *Georgia Institute of Technology*
2019

Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework
Robert J. Moore, *IBM Research–Almaden*
Raphael Arar, *IBM Research–Almaden*
2019

Heterogeneous Computing: Hardware and Software Perspectives
Mohamed Zahran, *New York University*
2019

Hardness of Approximation Between P and NP
Aviad Rubinstein, *Stanford University*
2019

The Handbook of Multimodal-Multisensor Interfaces, Volume 3: Language Processing, Software, Commercialization, and Emerging Directions
Editors: Sharon Oviatt, *Monash University*
Björn Schuller, *Imperial College London and University of Augsburg*
Philip R. Cohen, *Monash University*
Daniel Sonntag, *German Research Center for Artificial Intelligence (DFKI)*
Gerasimos Potamianos, *University of Thessaly*
Antonio Kruger, *Saarland University and German Research Center for Artificial Intelligence (DFKI)*
2019

Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker
Editor: Michael L. Brodie, *Massachusetts Institute of Technology*
2018

The Handbook of Multimodal-Multisensor Interfaces, Volume 2: Signal Processing, Architectures, and Detection of Emotion and Cognition
Editors: Sharon Oviatt, *Monash University*
Björn Schuller, *University of Augsburg and Imperial College London*
Philip R. Cohen, *Monash University*
Daniel Sonntag, *German Research Center for Artificial Intelligence (DFKI)*
Gerasimos Potamianos, *University of Thessaly*
Antonio Kruger, *Saarland University and German Research Center for Artificial Intelligence (DFKI)*
2018

Declarative Logic Programming: Theory, Systems, and Applications
Editors: Michael Kifer, *Stony Brook University*
Yanhong Annie Liu, *Stony Brook University*
2018

The Sparse Fourier Transform: Theory and Practice
Haitham Hassanieh, *University of Illinois at Urbana-Champaign*
2018

The Continuing Arms Race: Code-Reuse Attacks and Defenses
Editors: Per Larsen, *Immunant, Inc.*
Ahmad-Reza Sadeghi, *Technische Universität Darmstadt*
2018

Frontiers of Multimedia Research
Editor: Shih-Fu Chang, *Columbia University*
2018

Shared-Memory Parallelism Can Be Simple, Fast, and Scalable
Julian Shun, *University of California, Berkeley*
2017

Computational Prediction of Protein Complexes from Protein Interaction Networks
Sriganesh Srihari, *The University of Queensland Institute for Molecular Bioscience*
Chern Han Yong, *Duke-National University of Singapore Medical School*
Limsoon Wong, *National University of Singapore*
2017

The Handbook of Multimodal-Multisensor Interfaces, Volume 1: Foundations, User Modeling, and Common Modality Combinations
Editors: Sharon Oviatt, *Incaa Designs*
Björn Schuller, *University of Passau and Imperial College London*
Philip R. Cohen, *Voicebox Technologies*
Daniel Sonntag, *German Research Center for Artificial Intelligence (DFKI)*
Gerasimos Potamianos, *University of Thessaly*
Antonio Kruğer, *Saarland University and German Research Center for Artificial Intelligence (DFKI)*
2017

Communities of Computing: Computer Science and Society in the ACM
Thomas J. Misa, Editor, *University of Minnesota*
2017

Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining
ChengXiang Zhai, *University of Illinois at Urbana–Champaign*
Sean Massung, *University of Illinois at Urbana–Champaign*
2016

An Architecture for Fast and General Data Processing on Large Clusters
Matei Zaharia, *Stanford University*
2016

Reactive Internet Programming: State Chart XML in Action
Franck Barbier, *University of Pau, France*
2016

Verified Functional Programming in Agda
Aaron Stump, *The University of Iowa*
2016

The VR Book: Human-Centered Design for Virtual Reality
Jason Jerald, *NextGen Interactions*
2016

Ada's Legacy: Cultures of Computing from the Victorian to the Digital Age
Robin Hammerman, *Stevens Institute of Technology*
Andrew L. Russell, *Stevens Institute of Technology*
2016

Edmund Berkeley and the Social Responsibility of Computer Professionals
Bernadette Longo, *New Jersey Institute of Technology*
2015

Candidate Multilinear Maps
Sanjam Garg, *University of California, Berkeley*
2015

Smarter Than Their Machines: Oral Histories of Pioneers in Interactive Computing
John Cullinane, *Northeastern University; Mossavar-Rahmani Center for Business and Government, John F. Kennedy School of Government, Harvard University*
2015

A Framework for Scientific Discovery through Video Games
Seth Cooper, *University of Washington*
2014

Trust Extension as a Mechanism for Secure Code Execution on Commodity Computers
Bryan Jeffrey Parno, *Microsoft Research*
2014

Embracing Interference in Wireless Systems
Shyamnath Gollakota, *University of Washington*
2014

Code Nation

Personal Computing and the Learn to Program Movement in America

Michael J. Halvorson

Pacific Lutheran University

ACM Books #32

Code Nation: Personal Computing and the Learn to Program Movement in America
Michael J. Halvorson

books.acm.org
http://books.acm.org

ISBN: 978-1-4503-7758-4 hardcover
ISBN: 978-1-4503-7757-7 paperback
ISBN: 978-1-4503-7756-0 EPUB
ISBN: 978-1-4503-7755-3 eBook

Series ISSN: 2374-6769 print 2374-6777 electronic

DOIs:

10.1145/3368274 Book
10.1145/3368274.3368275 Acknowledgments
10.1145/3368274.3368276 Chapter 1
10.1145/3368274.3368277 Chapter 2
10.1145/3368274.3368278 Chapter 3
10.1145/3368274.3368279 Chapter 4
10.1145/3368274.3368280 Chapter 5
10.1145/3368274.3368281 Chapter 6
10.1145/3368274.3368282 Chapter 7
10.1145/3368274.3368283 Chapter 8
10.1145/3368274.3368284 Chapter 9
10.1145/3368274.3368285 Chapter 10
10.1145/3368274.3368286 Chapter 11
10.1145/3368274.3368287 Afterword
10.1145/3368274.3368288 Index

A publication in the ACM Books series, #32
Editor in Chief: Sanjiva Prasad, *IIT Delhi*
Area Editor: Thomas Misa, *University of Minnesota*

This book was typeset in Arnhem Pro 10/14 and Flama using LuaTeX.
Cover image by vgajic/Collection E+ via Getty images

First Edition

10 9 8 7 6 5 4 3 2 1

Contents

Acknowledgments

I would like to thank the many friends, colleagues, and supporting institutions that have helped me bring this book to you. As all historians know, writers and researchers have many debts, and only some of them are repaid in the acknowledgments. As I complete this project, I am especially aware of the creative people and research institutions that have supported the *Code Nation* project over the past 5 years. I am also deeply aware of the many teachers and mentors that I have had the privilege to work with in a long career related to computing and higher education. This book is dedicated to you all, with my heartfelt thanks.

At ACM Books, I would like to thank Thomas Misa, who initiated this project and encouraged me as I worked on the early chapters. Tamer Ozsu and Achi Dosanjh provided helpful guidance as the project took shape and made its way into the publishing system at the venerable Association for Computing Machinery (ACM). Also at the ACM, I would like to thank Barbara Ryan, for her help with permissions, and Bernadette Shade, for her support with print production and graphics. Kim Halvorson sized and edited many of the images included before they entered production. Finally, I am grateful to Karen Grace for her careful editorial work as this book made its way through the publishing system.

At Pacific Lutheran University (PLU), I would like to thank the Provost and President's Offices for taking an interest in the *Code Nation* project and offering a sabbatical to support the writing of the first chapters. I am very grateful to the talented staff of the Mordvedt Library, who helped me locate many obscure books and magazines through interlibrary loans. My history, computer science, and innovation studies colleagues have been welcoming conversation partners throughout the project. Damian Alessandro contributed research to Chapter 2, and we shared many enjoyable chats about the early years of Apple Computer and counterculture activities in the San Francisco Bay Area. Michael Schleeter in the Department of Philosophy has been a great teaching partner, and I have benefited greatly from his wisdom about ethics and technology. Finally, Dale and Jolita Benson have been very

generous supporters of this book from its beginning, and I appreciate their many contributions to the Business and Economic History program at PLU.

At the University of Washington, Steven Pfaff in the Department of Sociology offered many helpful comments, especially ideas related to "technology enthusiasm" and using computing mythologies as a way to tease out the subtle relationships between technology initiatives and popular movements in the U.S. The staff of the Engineering Library helped me locate many obscure computer books and magazines.

At the Computer History Museum in Fremont, California, Sara Lott helped me to locate and attain permissions for many of the images included in this book. Sydney Olson welcomed me for several on-site visits to the museum's fantastic archive and reading room, and she tracked down materials in the collection when I had no idea where to look.

At Code.org, I am grateful for the help of Hadi Partovi, Alice Steinglass, and Lian Swanson, who connected me with resources about the Hour of Code campaign and the group's promising strategies for expanding access to computer science education in schools.

Searching for images and materials took me to numerous corporations, institutions, and databases in the U.S. I am deeply grateful to the following locations for sharing their resources with me: the Alabama Department of Archives and History, Apple Computer, the ACM, Code.org, Dartmouth College, DEC, Getty Images, Hewlett-Packard, IBM, John Wiley & Sons, Microsoft, Macworld, O'Reilly Media, PC/Computing, PC Magazine, PC World, Pearson, Penguin Random House, Springer Nature, University of Minnesota Libraries/Charles Babbage Institute, University of Washington Libraries, and Ziff Davis.

Oral history played an important role in this project, and I would like to thank the following people who shared interviews, email correspondence, photos, and/or printed materials with me: Renzhi Cao, Ray Duncan, Kevin Eagan, Lee Felsenstein, John Froschauer, Ken Goffman, Dan Gookin, Kim Halvorson, Dean Holmes, Alison Bailey Kennedy, André LaMothe, Dail Magee, Jr., Theresa Mannix, Robert M. McClure, Bart Nagel, Ted Nelson, Larry Osterman, Charles Petzold, Brian Randell, Jeffrey Richter, David Rygmyr, Megan Sheppard, Amy Stevenson, Patty Stonesifer, Mitchell Waite, Jim Warren, and Van Wolverton.

Finally, I reserve my deepest gratitude for my family, who supported me in untold ways as this project made its way from a collection of curious ideas to a completed book about the history of programming and personal computing. My sons Henry and Felix grew up hearing tales about early personal computers, and we have often chatted about software, music, and computer gaming. Kim has taken it all in stride because she was there for most of it, and often sees what other people miss. I am immeasurably grateful for my life with you.

PART I

LEARNING TO CODE

1 How Important is Programming?

"To understand computers is to know about programming. The world is divided... into people who have written a program and people who have not."

Ted Nelson, *Computer Lib/Dream Machines* (1974)

How important is it for *you* to learn to program a computer?

Since the introduction of the first digital electronic computers in the 1940s, people have answered this question in surprisingly different ways.

During the first wave of commercial computing—in the 1950s and 1960s, when large and expensive mainframe computers filled entire rooms—the standard advice was that only a *limited* number of specialists would be needed to program computers using simple input devices like switches, punched cards, and paper tape. Even during the so-called "golden age" of corporate computing in America—the mid- to late 1960s—it was still unclear how many programming technicians would be needed to support the rapid computerization of the nation's business, military, and commercial operations. For a while, some experts thought that well-designed computer systems might eventually program themselves, requiring only a handful of attentive managers to keep an eye on the machines.

By the late 1970s and early 1980s, however, the rapid emergence of personal computers (PCs), and continuing shortages of computer professionals, shifted popular thinking on the issue. When consumers began to adopt low-priced PCs like the Apple II (1977), the IBM PC (1981), and the Commodore 64 (1982) by the millions, it seemed obvious that ground-breaking changes were afoot. The "PC Revolution" opened up new frontiers, employed tens of thousands of people, and (according to some enthusiasts) demanded new approaches to computer literacy. As Ted Nelson, a prolific inventor and computing advocate wrote, "You can and must understand computers NOW!" On learning to program computers, Nelson energetically compared programming to another American obsession—driving an

Figure 1.1 **American school children experiment with computer programming using teletype machines (1970s).** (Courtesy of the Computer History Museum)

automobile. "If you've never written a program, it's like never having driven a car," Nelson instructed. "You may get the general idea, but you may have little clear sense of the options, dangers, constraints, possibilities, difficulties, limitations, and complications."[1]

Ted Nelson was not alone. By the late 1970s, scores of programming advocates recommended that people of all ages learn to code as a way of understanding what the world's most intriguing devices were capable of. *Computer programming*—a process of formulating a problem for the computer to solve, writing instructions in a given computer language, loading instructions into the computer's memory, running the program, and correcting errors—had emerged as a major late-night pastime and (for some) a promising profession. In response to the mandate of Nelson and others, a surge of interest in programming developed, and the number of people who could write at least elementary programs grew from several thousand in

1. Ted Nelson, *Computer Lib Dream Machines* (Self-published, 1974; Microsoft Press revised edition, 1987), 40.

the early 1950s into millions by the early 1980s. (See Figure 1.1.) This sea change in computational literacy encouraged the widespread adoption of computers, boosted the global economy, and shaped the contours of the modern information age.

1.1 Programming Culture

This book is about the rise of computer programmers and the emerging social, technical, and commercial worldview that I call *programming culture*, which took a distinctive form during the early decades of microcomputers and personal computing, *c.* 1970–1995. It is a popular history of coding that explores the experiences of novice computer users, tinkerers, hackers, and power users, as well as the ideals and aspirations of computer scientists, educators, engineers, and entrepreneurs. A central part of this story is the *learn-to-program movement*, which germinated in government and university labs during the 1950s, gained momentum through counterculture experiments in the 1960s and early 1970s, became a broad-based educational agenda in the late 1970s and early 1980s, and was transformed by commercialization practices in the 1990s and 2000s. The learn-to-program movement sought to make computers more understandable, imprint useful technical skills, establish shared values, build virtual communities, and offer economic opportunities for technology enthusiasts. The movement also supported user communities, schools, and emerging commercial industries, many of which benefited from the utility and connectivity provided by digital electronic computers.

The learn-to-program movement had its ups and downs, but eventually set the stage for 21st century expressions of computational literacy, such as the Hour of Code, YouTube and Lynda courseware, certification programs, coding boot camps, and university degrees in disciplines such as computer science, software engineering, information technology, artificial intelligence, and (most recently) human–computer interaction. As the title of this book suggests, the learn-to-program movement fostered a groundswell of popular support for computing culture in America, resulting in what I call a *Code Nation*—a globally-connected society that is saturated with computer technology and enchanted by software and its creation.

The learn-to-program movement (or more broadly, the software-maker movement) has inspired both disciples and critics. It has evolved over time and its advocates have traversed numerous professional boundaries and cultural institutions. The movement is historically distinct but also follows the patterns and rhythms of earlier socio-technical transformations, including the introduction of steam-powered technologies during the Industrial Revolution, the electrification of American businesses and homes, and the production of automobiles and "car culture" in the early 20th century.

Borrowing terminology from information science and the history of technology, the learn-to-program movement is identifiable as part of the "diffusion" and "domestication" phases that take place when a successful new technology is spread or "propagated" across society.[2] Scholars from the field of business and economic history also recognize this transition as a key period in which a new discovery or invention is widely adopted and made useful for the general public, resulting in new consumer behaviors and potential changes in the way that a market or the broader economy functions.[3] To achieve wide-spread diffusion, the movement often benefits from sustaining ideologies that strengthen the allegiance of followers and justify the time, resources, and commitment that are necessary for the movement's success.

Beyond hopes for material gain, America's expanding programming culture can also be viewed as a manifestation of the deep and abiding cultural tendency that many describe as "technological enthusiasm."[4] Technological enthusiasm is an upbeat, optimistic appraisal of new technical systems that not only stoke the engines of capitalism, but provide access to the ideals embedded in what is known as the American Project and the American Dream. The publishers of PC software systems readily participated in this vision, as each wave of entrepreneur–engineer strived to improve their software, best their rivals, and boost the productivity of corporations and the general public. By the 1980s, software creation had taken the form of a consensus ideology that united many Americans in a common, accessible dream of a better future through computing. As I will discuss in Chapter 2, this enthusiasm brought additional computing mythologies to the fore, and their collective use contributed to the positive view that American's held about PCs and software in the years to come.

2. See *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, edited by Margaret S. Elliott and Kenneth L. Kraemer (Medford, NJ: Information Today, Inc., 2008).

3. For a discussion of the phases that take place when a new consumer technology is introduced, see Joseph J. Corn, *User Unfriendly: Consumer Struggles with Personal Technologies, from Clocks and Sewing Machines to Cars and Computers* (Baltimore, MD: Johns Hopkins University Press, 2011). Also useful is Claude S. Fischer, *America Calling: A Social History of the Telephone to 1940* (Berkeley: University of California Press, 1994); and the essay collection *Does Technology Drive History? The Dilemma of Technological Determinism*, eds. Merritt Roe Smith and Leo Marx (Cambridge, MA: The MIT Press, 1994).

4. See Thomas P. Hughes, *American Genesis: A Century of Invention and Technological Enthusiasm, 1870–1970*, Second Edition (Chicago: University of Chicago Press, 2004); David A. Hounshell, *From the American System to Mass Production, 1800–1932: The Development of Manufacturing Technology in the United States* (Baltimore, MD: The Johns Hopkins University Press, 1984).

1.2 Learning a Language

By the late 1960s, programming emerged from America's research labs and government institutions to have a direct influence on universities, primary and secondary schools (K-12 in the U.S.), and the nation's businesses. But what type of mental activity did programming entail? How should students take their first steps when learning to program a computer? In search of an analogy, some specialists suggested that learning to program was a bit like learning to read or speak in a foreign language. Utilizing the comparison, some educators pressed for the inclusion of computer languages in their school's curriculum. Rather than taking a year or two of a spoken language, such as Spanish or German, a few innovative programs offered courses in computer language instruction, including FORTRAN, Logo, BASIC, and Pascal.

School administrators eager to provide practical job training (and to mollify prospective students and their parents) broadened the definition of "foreign language" to include instruction in computer languages, algorithms, and database management. The popular press advocated for coding instruction in newspapers and special reports, and computer book and magazine publishers released hundreds of titles to help students build simple applications for time-sharing systems and the first PCs.

No one argued that computer languages were the *same* as human languages, of course. But programming advocates pointed to the many parallels that they observed in the structure of spoken and computer grammars, and to the ways that basic logic and reasoning were gradually introduced to students. Instruction in programming seemed to permit access to the private world of a computer and its "brain" or central processing unit (CPU). Programming was also portrayed as a valuable exercise in logical thinking and problem solving. It was a mental activity that provided a conceptual introduction to how computers worked, as well as a deep dive into logic and syntax. For all these reasons, computer literacy advocates recommended that those who planned to use computers in the future should learn to code as soon as possible. "Even if you don't write programs yourself," Ted Nelson advised in 1974, "you may have to work with people who do."[5]

In the early years of the electronic computer revolution, it was the imposing image of the new *machines* that seemed to fascinate the public. The physicality of mainframe computers was reinforced by images of large devices whirring and blinking in popular films such as *Desk Set* (1957), *2001: A Space Odyssey* (1968), *Colossus: The Forbin Project* (1970), *Logan's Run* (1976), and *War Games* (1983). As computers became more reliable and better understood, however, the focus of

5. Nelson, *Computer Lib*, 43.

popular attention turned away from computing machinery to *software*, the programs that ran on computers, and the coding experts who wrote code in high-level languages like FORTRAN, COBOL, BASIC, and C. The computer industry went through many transitions in the 1960s and 1970s, adding minicomputers and other special-purpose machines. Gradually, the attention of the computing community shifted from scientific and military systems to the application software that powered new types of businesses and helped them manage information.

By the 1980s and 1990s, it became apparent that there were *not enough qualified programmers* to design, build, and maintain the software systems needed in the U.S. as the country expanded its computational interests into new areas. Although the academic discipline of computer science had taken shape in America's colleges and universities, these programs could not graduate enough scientists and engineers to satisfy the industry's needs. The situation was much the same in the rest of the computerized world, in schools and markets stretching from Europe to Asia. Journalist Clive Thompson has written about it this way: "If you look at the history of the world, there are points in time when different professions become suddenly crucial, and their practitioners suddenly powerful. The world abruptly needs and rewards their particular set of skills."[6] Computer programmers suddenly became this influential group.

The "big bang" of software construction that took place in the 1970s created waves of demand for qualified programmers that continue to expand up to the present. Even in the Internet age—when learning to manage websites, write blogs, and use social media tools has taken on great importance—learning to code has not lost its appeal. As this book goes to press, the leaders of technology companies such as Amazon, Google, Facebook, Apple, and Microsoft regularly complain to Congress that the U.S. does not have enough qualified software developers to meet its needs. According to these advocates, a special exemption is needed in our national immigration policies to allow more foreign high-tech workers into the U.S. to satisfy the demand for software developers and associated fields, such as hardware engineering, artificial intelligence, data mining, computer security, user interface design, audio engineering, cloud computing, product testing and verification, technical writing, product support, project management, and related fields. Programmers have become the lifeblood of our technical society.

1.3 New Ways of Thinking

Calls to learn coding techniques abound now from the leaders of our digital economy. So, too, are warnings that if a group does *not* heed the call, they will miss

6. Clive Thompson, *Coders: The Making of a New Tribe and the Remaking of the World* (New York: Penguin Press, 2019), 11.

out on all or part of what the global digital economy has to offer. But where did this urgency to learn programming come from? What has motivated schools and non-profit organizations to devote so many resources to preparing instructions for a computer? When did programming literacy emerge as a national priority? And what were the early experiences of programmers as they tinkered with mainframes, minicomputers, and the first microcomputers? How is this story connected to the development of successful platforms such as CP/M, MS-DOS, Microsoft Windows, the Apple Macintosh, and Unix-based systems?

Whether past or present, the invitation to become a software maker is an invitation to join a distinctive community within our global society and economy. This computing subculture was founded by a small group of research scientists and academics, but it has expanded into a diverse assortment of hobbyists, students, gamers, artists, musicians, hackers, engineers, career professionals, and part-time workers. Although each of these groups is distinct in socio-economic terms, there is discernable common ground in their understanding of computers and computing technology. Computer programmers share a basic orientation to the world that is shared, despite the differences that they experience in relation to hardware and software systems, learning tools, and historical context.

As a thought experiment, imagine that each subgroup within the programming collective can be conceived of as a concentric circle. In such a model of our programming culture, the entire assortment of circles would be drawn in close proximity to one another, and most of the circles would have points of intersection and overlap. A shared exposure to computational thinking is the overlap, even if the programming languages that people learn (and the tools they write programs with) change over time. Some computer programmers may take up professional work, and others will remain as hobbyists or late-night hackers. Still others may learn programming skills as part of a journey that leads to other types of fruitful work. Despite the differences, and there will be many, the entire set of circles is a model of our nation's programming culture.[7]

7. Georg Simmel first developed the idea of "cross-cutting social circles" to discuss how different groups meet at points of common interest, dispute, or compromise. See Georg Simmel, *Conflict and The Web of Group-Affiliations*, trans. Kurt H. Wolff and Reinhard Bendix, respectively (Glencoe, IL, 1955, original Berlin, 1908). For additional studies in the history of technology that have influenced my approach, see Joseph J. Corn, ed., *Imagining Tomorrow: History, Technology, and the American Future* (Cambridge, MA: The MIT Press, 1986); David E. Nye, *Narratives and Spaces: Technology and the Construction of American Culture* (Exeter, UK: University of Exeter Press, 1997); Nina Lerman, Arwen Mohun, and Ruth Oldenziel, "The shoulders we stand on and the view from here: Historiography and directions for research," *Technology and Culture* 38 (1997): 9–30; David E. Nye, *Consuming Power: A Social History of American Energies* (Cambridge, MA: The MIT Press, 1998); Joseph J. Corn, *The Winged Gospel: America's Romance with Aviation* (Baltimore, MD: Johns Hopkins University Press, 2002); Greg Downey, "Commentary: The Place of Labor in the

The call to join ranks with computer programmers is not just an invitation to new ways of thinking (learning computational logic) and new consumer behaviors (buying software and a programming primer), it is also a call to new social relationships, to new ways of seeing and knowing, and to participating in new personal and professional networks. The programming circles that collectively shape America's technical identity are as much expressions of a distinct subculture as are the ideas and values of Impressionist artists and their admirers in *Fin-di-siècle* Paris or jazz musicians and their fans during the Swing Era in New York City.

As a social historian with interests in the history of technology, business, and education, I am curious about the experiences of today's programmers and software creators, and where they received their training, inspiration, and cultural worldviews. (See Figure 1.2.) Although the Internet era has contributed much to the behaviors and identity of contemporary software developers, the core skills and thought patterns of modern programmers were influenced by even earlier commitments and achievements. These included the proliferation of high-level languages in the 1950s, the introduction of software engineering techniques in the 1960s, the idealism of educators, entrepreneurs, and authors in the 1970s and 1980s, and the diffusion of commercial programming techniques in the 1990s and 2000s.

My argument is that the learn-to-program movement gained momentum through each of these important transitions, as programmers, authors, and entrepreneurs created pathways through which Americans might learn programming skills and the fine-points of creating software for specific platforms. Computer book authors, magazine publishers, and influential programmer/educators played important, if overlooked, roles in the diffusion of these new skills. By establishing an ideological connection to the computer literacy movement, programmer/educators established a framework that made computer programming feel important, rewarding, and attached to the rituals of citizenship and corporate belonging. The learn-to-program movement took shape through the efforts of many unsung heroes, both women and men, and one of my goals with this book is to reacquaint historians and programmers with a cast of interesting actors and protagonists who have been left out of recent narratives. Part of this work involves using visual sources to

History of Information-Technology Revolutions," in *Uncovering Labour in Information Revolutions, 1750–2000 (International Review of Social History),* vol. 38 Supplement 11 (2003), 225–261; Lisa Gitelman, *Always Already New: Media, History, and the Data of Culture* (Cambridge, MA: The MIT Press, 2008); and Christopher Tozzi, *For Fun and Profit: A History of the Free and Open Source Software Revolution* (Cambridge, MA: The MIT Press, 2017).

Figure 1.2 **A middle school student learns computational thinking in a programming camp sponsored by the Tacoma/South Puget Sound MESA organization.** (Photo: Joshua Wiersma/Pacific Lutheran University)

unpack the social context of historic computing environments. (See Figure 1.3.) I will profile social reformers, writers, teachers, tinkerers, entrepreneurs, and hackers, as well as computer scientists, students, engineers, and the leaders of America's computing societies, including the Association for Computer Machinery (ACM). Predictably, most of the programmers that we meet will be members of more than one social or professional group.

To get a sense for the magnitude of the sea change that took place, consider some basic demographics. In 1957, there were approximately 15,000 computer programmers employed in the U.S., a figure that accounts for approximately 80% of the world's programmers active that year. The work of the first computing pioneers involved building and maintaining military systems, designing algorithms for scientific research, tracking census data, and implementing data-processing schemes for government bureaus and corporations.

In 2000, there were approximately 9 million professional programmers worldwide, with millions more who had been exposed to coding concepts as part of

Figure 1.3 **Three men and two women gather for a meeting near an IBM 370 Model 138 Computer System in Berkeley, California. IBM's 1976 publicity photo emphasizes the value of teamwork and the extensive documentation that was prepared for programmers and administrators.** (Courtesy of the Computer History Museum)

their school curriculum or other experiences.[8] In addition to steady growth in military and scientific computing, the expanding digital economy has brought new opportunities for computer programmers in the fields of consumer software, video game programming, artificial intelligence, information publishing, digital communications, education, art, music, entertainment, medicine, and other areas that benefit from the use of computers.

The rising tide of opportunity for software developers has continued up to the present. In 2014, there were approximately 18.5 million software developers in the world, of which 11 million can be considered professional programmers and

8. Steve Lohr, *Go To: The Story of the Math Majors, Bridge Players, Engineers, Chess Wizards, Maverick Scientists, and Iconoclasts—The Programmers Who Created the Software Revolution* (New York: Basic Books, 2001), 6–7.

7.5 million can be considered as hobbyists.[9] Many programmers create or maintain software as part of their regular employment, while others write code for non-profit organizations that they support, and still others program at school, for recreation, or as an aspect of their personal or professional development.

1.4 Equity and Access

Despite the bright economic outlook for software developers, there are still numerous challenges in bringing programming proficiencies to the general population. In reality, only a small subset of the people who use computers actually go on to learn something about computational thinking or software development. Our modern economy requires many important job skills and personal investments. Considering the costs and the effort required, does it really matter who learns to program and who does not?

In the book *Stuck in the Shallow End: Education, Race, and Computing*, Jane Margolis et al. argue the "who" that learns to use technology matters a great deal, and that America has suffered throughout its history from inequities in access to computing.[10] Their research indicates that African-American and Latino children are *much less likely* to receive technology training in American schools than white or Asian children. When scholars analyze gender disparities and later professional outcomes, they find that only two out of ten information technology (IT) professionals are women in the current U.S. workforce.

Margolis and her contributors offer convincing evidence that the characteristics of programming culture matter tremendously to those who enter the subculture and to those who thrive in it (or recede from view). Understanding the long history of the learn-to-program movement and its cultural commitments and values reveals much about how people have interacted with computers in the past, and how we might expand computing opportunities in the future. Yasmin Kafai and Quinn Burke describe the challenge before us as working to better support "computational participation" in our schools and professional environments. In their important book, *Connected Code*, they recommend that thought leaders

9. International Data Corporation, *2014 Worldwide Software Developer and ICT-Skilled Worker Estimates* (Framingham, MA: International Data Corporation, 2014).

10. Jane Margolis, Rachel Estrella, Joanna Goode, Jennifer Jellison Holme, and Kimberly Nao, *Stuck in the Shallow End: Education, Race, and Computing*, Updated Edition (Cambridge, MA: The MIT Press, 2017). See also J. Margolis, J. Goode, and K. Binning, "Exploring computer science: active learning for broadening participation in computing," *Computing Research News* 27, no. 9 (October 2015).

Figure 1.4 **ACM member Professor Renzhi Cao teaches computer science to local middle school students in Tacoma, WA. Early engagement and outreach related to computational thinking has become a standard practice in many high-tech communities.** (Photo: John Froschauer / Pacific Lutheran University)

shape technology-centered cultures carefully, ensuring that *all* participants feel welcomed and included.[11] (See Figure 1.4.)

One important outgrowth of this research has been the rise of not-for-profit organizations that teach young people how to program, including Code.org, Black Girls Code, Girls Who Code, Native Girls Code, and The Hidden Genius Project. At the high school level, many organizations focus on introducing programming concepts and preparing students to take the College Board's AP Computer Science Principles examination. I evaluate the work of Code.org and the Hour of Code movement in the Afterword for this book. As a preview, I note here that Code.org has completed over 720 million introductory programming sessions since the organization began in 2013, with 46% female and 48% underrepresented minorities currently

11. Yasmin Kafai and Quinn Burke, *Connected Code: Why Children Need to Learn Programming* (Cambridge, MA: The MIT Press, 2016).

using the group's courseware.[12] These figures are clearly impressive, and they show one way that creative thinking and industry partnerships can support an education system that is struggling to find resources and leadership. However, the new initiatives were not created from whole cloth, but are simply the latest manifestations of a long programming literacy movement that has a fascinating history and is now being scaled to meet global needs. Just as in the past, there is an ongoing debate about the efficacy of wide-ranging computer literacy programs and the best way to deliver them.[13]

Importantly, this conversation about equity and access is connected to ethical considerations, and it will only move forward with input from many academic and industry partners. Our world is increasingly dependent on computers and technology, and it is imperative that we work together to understand the characteristics of technical communities and how they shape our hearts and minds. Computational thinking courses are among the most interesting places to attempt this work.

1.5 Personal Connections

I have wanted to write this book for a long time because I am fascinated with software development. PCs were an important starting place for me during my teenage years, and I first learned to write computer code on early microcomputers and PCs. Like many people of my age and social context, my first experiments with home electronics took place in the family rec room during the late 1970s. My extended family bought a Tandy TRS-80 and an Atari video computer system, and the young people in our circles used them to play video games like *Pong* and *Missile Command*. A bit later, I experimented with an early IBM Personal Computer when it was released in August 1981, just weeks before I entered college at Pacific Lutheran University (PLU) in Tacoma, Washington. I took an introductory computer programming course and declared as a Computer Science major at PLU, deferring my interests in history and education for graduate school. I learned BASIC, Pascal, C, and assembly language programming on the university's Digital Equipment Corporation (DEC) VAX 11-780 and DEC PDP-11 minicomputers. I also studied mathematics, data structures, algorithms, operating systems, digital logic, computer architecture, computer graphics, and networking theory. In 1985, I

12. See "Code.org 2018 Annual Report," February 12, 2019, 3. https://code.org/files/annual-report-2018.pdf. Accessed August 9, 2019.

13. For a summary of the current concerns and priorities in the computational literacy field, see Emmanuel Schanzer, Shriram Krishnamurthi, and Kathi Fisler, "Education: what does it mean for a computing curriculum to succeed?" *Communications of the ACM* 62, no. 5 (2019): 30–32.

Figure 1.5 **Michael Halvorson working in his office at Microsoft Corporation (1990).** (Photo courtesy of Michael Halvorson)

graduated from university and I was hired at Microsoft Corporation to work in one of their two Bellevue (Washington) office buildings, just before the company moved to its better-known Redmond campus. I was employee #850 in the rapidly expanding organization (see Figure 1.5), arriving when the best-selling products were MS-DOS, Microsoft Word for MS-DOS, and a few popular programming languages and development tools.

During my job interview at Microsoft, I was shown a testing (beta) version of Microsoft Windows 1.0. It was not very impressive at the time, but the new graphical operating environment for IBM PCs and compatibles would eventually become an exciting platform for many users, programmers, and commercial software publishers. My first work was at Microsoft Press, the book publishing division of Microsoft, founded by Bill Gates in 1983 to provide technical support for computer enthusiasts who were frustrated by the poor quality of software manuals. In the early days of personal computing, product documentation was often little more than printouts assembled in a three-ring binder, and there was not much in the way of computer-based help or training for PC users. From these humble beginnings, a

major publishing industry took shape. It came to include bestselling magazines like *PC Magazine*, *Macworld*, and *Compute!*, as well as the computer book publishers Howard W. Sams, O'Reilly, Osborne McGraw-Hill, Que, Microsoft Press, Sybex, and IDG Books.

Our work at Microsoft Press was to help self-taught programmers and those who used Microsoft's business applications to get the most out of their software. I edited books, worked with independent authors, attended industry trade shows, and (beginning in 1986) started writing do-it-yourself (DIY) computer books about using operating systems and programming languages. I was lucky that my university training required a healthy dose of the liberal arts along with my computing classes. Both fields of study prepared me to tackle substantial research and writing projects in the years to come, and they were valued in the book publishing division.

The learn-to-program movement was something that I saw first-hand while working with Microsoft's customers and authors. In particular, there were fascinating people to learn from at computer industry trade shows, especially COMDEX and Macworld Expo. (See Figure 1.6.) In 1989, I co-authored the book *Learn BASIC Now* with my colleague and friend, David Rygmyr, and the book was carefully edited by Megan Sheppard and Dale Magee, Jr. (also employees of Microsoft Press). Our programming courseware included a full-featured version of the Microsoft QuickBASIC Interpreter for MS-DOS on three 5.25" disks, and Bill Gates wrote a Foreword to the book recalling his personal connection to Altair BASIC and his interest in using BASIC as a unifying language across computing platforms. (See Chapter 5.)

Learn BASIC Now sold many copies and it was selected as a finalist for a national book award in the computer book "How To" category. Our self-study guide clearly intersected with the powerful demand for programming instruction, and the low-cost QuickBASIC Interpreter made the product relatively inexpensive for newcomers. Over the years, I wrote another 15 books about software development, mostly for self-taught programmers and those who wanted to learn the newest features of popular products like Microsoft Visual Basic or Microsoft Visual Studio. Through the books, I was actively connected to publishers, software development teams, user groups, academics, journalists, literary agents, and a wide range of computer users—many of whom would write or email us directly for help.

1.6 Manifestos of the Movement

Despite my positive interactions with new programmers, I gradually learned that I was only a small part of the third or fourth wave of technical writers who had spread the message about computational literacy and learning to code in the years since the

Figure 1.6 **An exhibitor badge from the COMDEX/Fall '90 trade show in Las Vegas, Nevada.** (Photo courtesy of Michael Halvorson)

introduction of the first computers. *Preparation of Programs for an Electronic Digital Computer* was published in 1951 by Maurice Wilkes, David Wheeler, and Stanley Gill to instruct readers on how to formulate machine code for the revolutionary EDSAC computer at the University of Cambridge.[14] Grace Mitchell, Daniel McCracken, and Elliott Organick also wrote creative programming primers for FORTRAN in the late 1950s and early 1960s, introducing non-specialists to programming.

In the era of time-sharing systems and early PCs, a new wave of programming advocates supported the movement. These were pioneers like Robert Albrecht and LeRoy Finkel, who participated in the People's Computer Company and the Homebrew Computer Club in Menlo Park, California. From the beginning, these visionaries understood that not only did people need to *buy computers* and start programming, but they needed to *learn how to program* through books, materials, and social interaction. These computing innovators wrote fascinating programs and produced several best-selling computer titles, but they have largely been neglected in the history of computing. A new book by Joy Lisi Rankin, *A People's History of Computing in the United States*, is an important exception to this lacuna, and Rankin demonstrates how Albrecht and his contemporaries inspired thousands of programmers to appreciate the benefits of BASIC.[15]

14. Maurice Wilkes, David Wheeler, and Stanley Gill, *Preparation of Programs for an Electronic Digital Computer* (Reading, MA: Addison-Wesley, 1951).

15. Joy Lisi Rankin, *A People's History of Computing in the United States* (Cambridge, MA: Harvard University Press, 2018), 68–69, 94–100.

Also important in the 1960s and 1970s were the pioneering efforts of the educational theorists Arthur Luehrmann, Seymour Papert, Cynthia Solomon, and Wally Feurzeig, all active in the computing hotbeds of Cambridge, Massachusetts and Greater Boston. Luehrmann coined the term "computer literacy" and encouraged students to learn structured programming with BASIC and Pascal. Papert, Solomon, and Feurzeig co-developed the Logo programming system at the Massachusetts Institute of Technology (MIT), and they wrote about its potential to teach computational thinking to children. Also, from the era of time-sharing systems, David Ahl, an early DEC employee, published tutorials that advocated for the use of computer games to teach programming concepts. My favorite of Ahl's titles is *101 BASIC Computer Games*, published by DEC in 1973. This book is filled with mimeographed program listings that Ahl received in the mail from BASIC users across the U.S. It was one of the first bestselling computer programming titles, selling tens of thousands of copies to novice computer users, hobbyists, academics, and working professionals.

Many of the earliest manifestos of the learn-to-program movement were sold out of VW vans and dusty boxes in computer clubs. However, this DIY world was also on the fringes of the professional software development community, which took its energy from debates within the nascent software engineering movement and the emerging discipline of computer science. The standard-bearers in this field created the computers, operating systems, and programming languages that would fuel the academic and commercial worlds of software development in the years to come. Readers learned about their important discoveries through conferences and influential computer books such as Donald Knuth, *The Art of Computer Programming* (1968 and later); Kathleen Jensen and Niklaus Wirth, *The Pascal User Manual and Report* (1971); Brian Kernighan and Dennis Ritchie, *The C Programming Language* (1978); and Rodnay Zaks, *Programming the Z80* (1979). Although these authors did not always publish programming primers, they helped experienced programmers understand the cadence of computer languages, taught people to devise data structures and algorithms, and explored the advanced features of operating systems and computer architecture. The introduction of professional and commercial programming practices is a crucial stage of the learn-to-program movement.

1.7 A New History of Personal Computing

Code Nation explores the social, technical, and commercial changes that took place in the U.S. as computer programming became a regular part of life for so many. The trials and triumphs of PC programmers are featured on these pages, as well

as the negative consequences that came to people who were denied the opportunity to code based on their location, gender, ethnicity, or economic circumstances. My emphasis is not on high-tech leadership strategies or the tactics that generated corporate wealth, but on the stories of lesser-known programmers, authors, academics, and entrepreneurs. Some were successful, and some have been mostly forgotten. But this is itself a lesson in the history of innovation, business, and technology.

To tell this tale, *Code Nation* presents a new history of personal computing in the U.S. I present a detailed analysis of early computer platforms, a discussion of important compilers and development tools, a "behind-the-scenes" look at application and operating-system programming, the origins of corporate and "enterprise" computing strategies, the rise of user's guides and computer books, and early attempts to market and sell PC software. Writing a fresh history of personal computing involves significant challenges, in part because the most recent storytelling emphasizes the roles that famous "pioneers" and "founders" have played in narratives about Silicon Valley, the Greater Boston area, and the Pacific Northwest. There has been no shortage of popular books about Apple Computer, Microsoft, Amazon, Google, and Facebook—usually emphasizing the rise of the stereotypical "computer nerds" to positions of wealth and influence in the companies that benefited from personal computing and Internet-based technologies.[16]

It is often difficult to move beyond these perspectives because of a curious lack of sources that document early personal computing and its broader impact on American society. Most of the earliest PC hardware and software companies have merged or gone out of business, leaving little in the way of historical materials to study. IBM is a noteworthy exception to this trend, recently releasing some of its materials to historians of computing.[17] But Apple Computer's corporate records have been carefully edited by their legal teams and are only partially available. Microsoft has also been reluctant to open its corporate archives to scholars and the general public. Beyond the personal narratives of former employees and product enthusiasts, how are historians to study the history of personal computing? What sources can we use to understand how corporate identities were shaped, hardware

16. An example of this work is Walter Isaacson, *The Innovators: How a Group of Hackers, Geniuses, and Geeks Created the Digital Revolution* (New York: Simon and Schuster, 2014). An intriguing new approach is Margaret O'Mara's history of Silicon Valley, which connects the technical and business development of the region to local and national politics. See Margaret O'Mara, *The Code: Silicon Valley and the Remaking of America* (New York: Penguin Press, 2019).

17. See James W. Cortada, *IBM: The Rise and Fall and Reinvention of a Global Icon* (Cambridge, MA: The MIT Press, 2019), especially chapter 14. Cortada was well positioned to write this history because he is a former IBM employee as well as a professional historian.

and software products were created, and whether computing initiatives succeeded or failed? Just as important, how did the *users* of PCs experience new products and come to understand their features? Can we assess how regular people accepted, accommodated, or rejected the plans and proposals of industry elites?

Code Nation proposes a publication-centered way of examining the early history of microcomputing and personal computing, from experiments with time-sharing systems, to the mail-order kits of early enthusiasts, to book and magazine publications for platforms like MS-DOS, the Apple Macintosh, Microsoft Windows, and Unix/Xenix. I evaluate the history of personal computing using hundreds of programming primers, textbooks, manuals, magazines, user's guides, and trade show catalogs from the early 1950s to the late 1990s. These neglected sources have allowed me to explore the challenges presented by the first PC systems, the content of computer literacy debates, the methodology of early programming primers, the strategies of successful (and unsuccessful) entrepreneurs and corporations, and the way that computing has impacted the daily life of Americans. To support this analysis, I include technical descriptions of hardware and software systems, code snippets from historic programming languages, the biographies of little-known programmers and entrepreneurs, and a product-based assessment of early hardware and software systems. I also present over 80 historic photographs selected from relevant archives, museums, corporations, and private collections.

I have learned that printed materials related to computers and software—once a common feature of many offices, homes, and schools—have been discarded at an alarming rate. When discussing the issue of "disappearing sources" with a local college librarian, I learned that older computer books and magazines are especially vulnerable to being categorized as *ephemera*, or transitory sources of information about outdated methods or technologies. (See Figure 1.7.) With new computer books and periodicals arriving on a monthly basis, and shrinking budgets, how important is it to maintain an historic collection of FORTRAN, BASIC, and C primers? Especially in locations where shelf space is at a premium? My source's questions are legitimate, of course. But the comment points out how vulnerable technical sources are to abandonment. "Often, they are simply recycled," my informant conceded.

But, if we cannot study issues like computer literacy in the past, how can we hope to evaluate it in the present?

For the purpose of this study, I was able to find many older computer books and periodicals in private collections, as well as the technical libraries of larger public universities. For example, I have spent many weeks in the engineering library at the University of Washington in Seattle, which has a good collection. I also found many books, newsletters, and software packages in the Computer History Museum

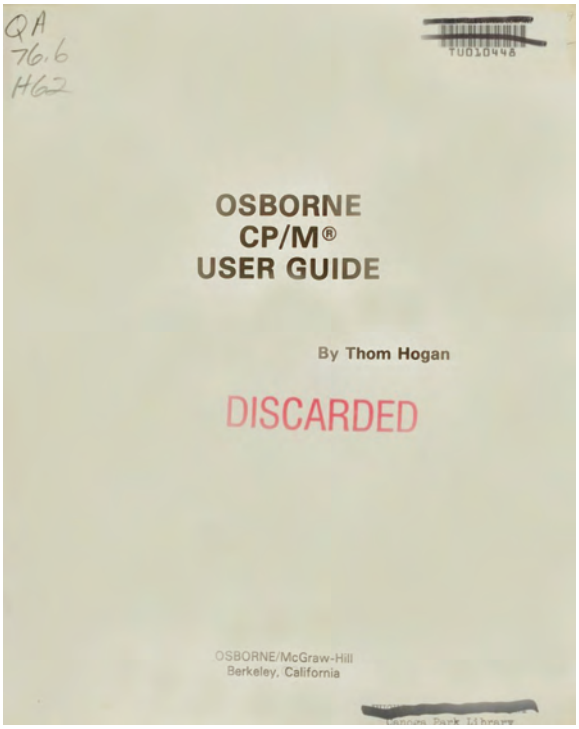


Figure 1.7 **The title page of Thom Hogan's *Osborne CP/M User Guide*. Published by Osborne/McGraw-Hill in 1981, this book was one of the most important operating system primers of the microcomputer era. Like many older computer publications, however, it has been widely discarded by libraries.** (Photo courtesy of Michael Halvorson)

in Fremont, California. But like the chapbooks and "street literature" of earlier eras, historic computer books and materials can easily be lost if historians are not sensitive to the many treasures that they contain. In particular, they reveal the teaching strategies used to introduce new technical systems, and the opinions and practices of regular people who are learning new technologies. I hope that this publication-centered approach will be of interest to future historians of computing. There are still many fascinating sources that slumber in our nation's technical collections.

I begin *Code Nation* with a comparative analysis that examines computing in the 1960s and 1970s, emphasizing the era's sense of crisis about how software was being created and its multilayered hopes for renewal. My survey presents four overlapping computing mythologies, each representing a different aspect of the period's professional, cultural, and technical traditions. These narratives introduce early advocates for software engineering practices, countercultural idealists who

promoted widespread access to tools, creative scholars from the emerging discipline of computer science, and the designers of the first personal computers. In the 1980s and 1990s, American programmers drew on many of these motifs, creating a worldview that bundled hopes, anxieties, and dreams about the new platforms.

2 Four Computing Mythologies

"Total learning expands when the range of spontaneous learning widens... and both liberty and discipline flower."

Ivan Illich, *Tools for Conviviality* (1973)

"In recent years, I have talked to a number of top industry researchers and implementors who are reluctant to hire computer science graduates at any level. They prefer to take engineers or mathematicians, even history majors, and teach them programming."

David Lorge Parnas, *Computer* (1990)[1]

When it comes to social movements, the groups that strive toward a common goal with a shared sense of purpose are often the most successful. The learn-to-program movement of the 1970s and 1980s fits this pattern, as do many of the recent computer literacy initiatives, including Code.org's Hour of Code. According to sociologists, the ideological beliefs that ground social movements act as a bulwark for striving organizations, strengthening the commitment of both leaders and members.[2] Ideological beliefs also help adherents imagine a new world order, and they justify the relatively high levels of personal investment and resources that social movements require. Ideologies set the expectations of a movement's believers, so that adherents can learn what the group is trying to accomplish and how they should propagate their beliefs. When the going gets tough, ideological commitments keep a social movement going.

1. Cited in David Lorge Parnas, "Education for computing professionals," *Computer* 23, no. 1 (Jan. 1990), 17–22.

2. Margaret S. Elliott and Kenneth L. Kraemer, "Computerization movements and the diffusion of technological innovations," in *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, edited by Margaret S. Elliott and Kenneth L. Kraemer (Medford, NJ: Information Today, Inc., 2008), 6.

This chapter explores four powerful ideologies that influenced America's burgeoning computer industry in the 1960s and 1970s, each influencing the learn-to-program movement in its own way. Although Parts II and III of this book narrate how Americans learned to code in the 1980s and 1990s, it was only through an awareness of earlier successes and failures that the microcomputing and personal computing movements took shape. Like other scholars, I choose to use the term *foundation myths* to describe the ideologies that influenced the computer industry as it emerged from research settings to become a major contributor to the U.S. economy. *Foundation myths are socially-constructed memories that can carry important historical and cultural information*. They act as social markers, transmitting ideas, beliefs, and worldviews to community members and future generations. Foundation myths summarize historical debates and scientific commitments. They often work subtly, employing the language of metaphor or ritual. In more recent times, computer-related myths are used to celebrate heroic founders and to marginalize illicit behavior. You can often spot these myths when subtle descriptors are used in histories and popular accounts, such as "pioneer," "entrepreneur," "evangelist," "guru," "hacker," and "cyberpunk."

Among the many possibilities, I have chosen four myths about computer technology and computer programmers to begin this book. I will draw connections between these ideologies and the learn-to-program movement in the chapters that follow. The first mythology is a belief in an ongoing period of crisis in the computer industry related to the complexity of computer systems and the pitfalls of commercial software development. Strongly held beliefs about this "crisis" emerged in the 1960s among software development communities, and it set the expectation that most large software projects would arrive late, over budget, and in poor shape. The second mythology is that the computer industry works best when it is driven by popular, democratic impulses and shared community values. This counterculture narrative emerged in the 1970s when a group of West Coast technology enthusiasts argued that using and programming computers should be an enlightening, communal experience. These values strongly shaped some segments of the computer industry, including the microcomputer community in California and authors of programming tutorials in the 1970s and 1980s.

The third mythology relates to a belief about the commitments of professors and administrators in the emerging discipline of computer science. In the U.S., many computer professionals came to believe that computer scientists were occupied primarily with theoretical problems related to computational logic, algorithms, and engineering principles, rather than the practical skills needed to implement projects in the computing industry. Although some academics *did* assume an aloof posture in relation to business computing, this stereotype was largely inaccurate, and it had important consequences for how professional programmers were trained

(or not trained) in the coming decades. Finally, there are several mythologies related to what is often called "the PC Revolution," a phrase that tries to capture the excitement surrounding the creation of the first stand-alone microcomputers and personal computers (PCs) in the 1970s and 1980s. This term draws attention to vital energies in the American computer industry, but it also tends to lionize the experience of PC users and entrepreneurs over professionals working in other areas of digital computing. After gently nudging aside this rhetoric, *Code Nation* proceeds by exploring how the microcomputer movement actually *did* contribute in important ways to the development of programming culture and the commercial software industry in America. We will investigate how this upward trajectory took place in waves or *stages*—from the time-sharing systems of the 1960s and 1970s, to the barebones microcomputers and PCs of the late 1970s and early 1980s, to the powerful graphical user interface (GUI) workstations of the late 1980s and early 1990s, to the corporate and enterprise computing systems of the late 1990s and early 2000s. These stages involved fascinating operating system platforms, including CP/M, Apple DOS, MS-DOS, the Apple Macintosh, Microsoft Windows, Unix/Xenix, OS/2, and Windows NT Server.

By giving powerful computing mythologies their due, we acknowledge the importance of cultural memories in the history of business and technology, including the problems that people encountered in the past, and the aspirations of users and programmers in the future. The learn-to-program movement succeeded in part because it wove together each of these mythologies, creatively transforming past memories into a shared vision of progress and human belonging. The movement's visionaries, authors, tinkers, and entrepreneurs deserve recognition for their contributions to the gradual computerization of society, a process with major cultural and economic consequences that is still underway.

We'll begin with a technical problem and a story.

2.1 The North Atlantic Treaty Organization (NATO) Conference on Software Engineering

In October 1968, there was a sense of crisis in the air.

Although this year has been described as one of the most turbulent in the 20th century, the turning point was not related to political or military disruptions, but to a crisis in the nascent field of software engineering. In fact, executives in North America and Europe had been sounding the alarm since the mid-1960s. Now that powerful mainframe computers were transforming the world's business and engineering systems, the software that drove these machines was taking on an oversized role in public life. Just weeks before Richard Nixon, Hubert Humphrey, and George Wallace faced off in the 1968 American Presidential Election, the world's engineers were worrying about computers and software.

To list a few of the problems, there was a perpetual shortage of programmers to create software for the new systems. These programs were often massive, stretching to tens of thousands—even millions—of lines of code in computer languages like COBOL, FORTRAN, and ALGOL. The code configured American military systems and corporate data processing tools—programs like the banking, billing, and reservation systems that proliferated in the late 1960s. However, good software developers were hard to find. There was no clear procedure for locating, hiring, and training the specialists needed to build and maintain the required systems.

The growing complexity of software also required robust management techniques to ensure that projects were completed on time and on budget, but neither outcome was very common. To make matters worse, the growth of professional organizations like the Association for Computing Machinery (ACM) found it difficult to improve programmer productivity or software quality. Although revenue was pouring in to successful American hardware manufacturers like IBM, Burroughs, and Digital Equipment Corporation (DEC), the budding software industry seemed undisciplined in its workflows, ill-prepared to expand, and in a perpetual state of disorder.

Much has been written about the "software crisis" of the late 1960s, and some have argued that "software engineering" was the wrong metaphor to address the problems.[3] But the dilemma was noted by many computer professionals around the world, from North America to Asia to Europe. The 1960s was a time of expansion, as organizations were drawn to utopian visions of mainframe and minicomputer technologies. But reassessments soon followed, and critics pointed to bloated software systems that were complex and error prone; programs designed for engineers with pocket protectors but not real people. The job performance of corporate software developers also came under fire. "When a computer programmer is good, he is very, very good," concluded one IBM study published in 1968. "But when he is bad, he is horrid."[4]

3. For a deeper look at the issues, see Janet Abbate, *Recoding Gender: Women's Changing Participation in Computing* (Cambridge, MA: The MIT Press, 2012), 97–111; Sandy Payette, "Hopper and Dijkstra: Crisis, revolution, and the future of programming," *IEEE Annals of the History of Computing* 36, no. 4 (2014): 64–73; Adam Barr, *The Problem with Software: Why Smart Engineers Write Bad Code* (Cambridge, MA: The MIT Press, 2018); Liesbeth De Mol and Giuseppe Primiero, eds., *Reflections on Programming Systems: Historical and Philosophical Aspects* (Cham, Switzerland: Springer, 2019).

4. Quoted in Hal Sackman, W. J. Erickson, and E. E. Grant, "Exploratory experimental studies comparing online and offline programming performance," *Communications of the ACM* 11, no. 1 (1968): 3–11.

Figure 2.1 **IBM Executives face the camera in front of a bank of IBM 729 magnetic tape drives, 1962.** (Photo by The LIFE Picture Collection via Getty Images/Getty Images)

The gender implications embedded in this statement are subtle, but important to catch. In 1968, approximately 88% of professional programmers in the U.S. were men. Although women made significant contributions to computing in the 1950s, programming work underwent a process of masculinization in Britain and America in the 1960s and 1970s.[5] As part of this transition, the cultural stereotypes about programming being a male activity increased, and the era's sources often associate programming with masculinity. The implications of the underrepresentation of women in computing will be examined in Chapters 3, 7, 8, and 10, which explore how Americans learned to code, and how new programmers negotiated for status in the communities that either welcomed or rejected them. Keep an eye on this complex issue; it surfaces in predictable but also unlikely settings. (See Figure 2.1.)

To address the global software crisis, the Science Committee of NATO sponsored a conference in October 1968, in the Garmisch-Partenkirchen district of West

5. See Abbate, *Recoding Gender*; Marie Hicks, *Programmed Inequality: How Britain Discarded Women Technologists and Lost Its Edge in Computing* (Cambridge, MA: The MIT Press, 2017); Nathan Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics and Technical Expertise* (Cambridge, MA: The MIT Press, 2010).

Germany (Bavaria). The organizers planned to discuss the design and production of computer software, the experiences of software users, and the persistent problem of meeting software schedules and budgets. The term "Software Engineering" was chosen as a framing title for the meetings, hinting at a proposed solution to the dilemma: The computer industry should infuse programming with theory and practice from the disciplines of science and engineering to address their problems. It was high time to demand structured approaches to design, coding practices, and testing that would improve reliability. If this did not happen soon, the complexity of software, and its concomitant unpredictability, would likely stifle the electronic computer revolution.

The 5-day conference was attended by 50 people from 11 countries, with a large contingent representing the U.S. All 50 registered conference participants were men, with a few women serving in support roles. Analyzing the conference materials, Janet Abbate sees the absence of the American Grace Murray Hopper as the most striking in terms of gender exclusion. In 1968, Hopper served as the director of programming languages and standards for the U.S. Navy, and she was an established luminary in the computing industry.[6]

The proceedings show that the attendees were mostly computing professionals who managed software teams or worked regularly with the users of software systems. There were manufacturers' representatives in attendance, as well as academics speaking for interested university faculties.[7] As Nathan Ensmenger wrote, the meeting marked a major shift in general perceptions about software construction.[8] After Garmisch, there was pressure on software developers to eschew craft and artisan approaches to programming and to adopt established engineering principles. The wild and woolly days of coding by trial and error were over. (Or so the organizers hoped.) Although some came to question the value of software engineering as a framing term, the methodology would have an important impact on programming culture in the coming decades.[9]

6. Abbate, *Recoding Gender*, 102–103.

7. Peter Naur and Brian Randell, eds., *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, Oct. 7–11, 1968* (Brussels: Scientific Affairs Division, North Atlantic Treaty Organization [NATO], 1969). I thank Brian Randell and Robert M. McClure for sharing important information with me about the conference via email and postal correspondence. See http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF. Accessed August 20, 2019.

8. Ensmenger, *The Computer Boys Take Over*, 196–197.

9. On the legacy of the 1968 conference, see Matti Tedre, *The Science of Computing: Shaping a Discipline* (Boca Raton, FL: CRC Press, 2015), 111–137. Also useful is Merlin Dorfman and Richard H. Thayer, eds., *Software Engineering* (Los Alamitos, CA: IEEE Computer Society Press, 1997).

Let's take a look at the problems software developers wrestled with in the 1960s. At that time, the dominant paradigm for software development revolved around a solitary (male) computer genius enigmatically holding court with a small team of assistants. The consensus at Garmisch was that this scenario needed to be replaced with a cohort of systematically-trained engineers, responsible to management, who practiced structural thinking and followed orders. (A visual model of this hierarchy and approach can be seen in Figure 2.2.) As Nathan Ensmenger summarized, "Software engineering promised to bring control and predictability to the traditionally undisciplined practices of software development."[10]

Historian Stuart Shapiro analyzed the legacy of this "engineering movement" as it gained momentum in the 1970s and 1980s. According to Shapiro, the program was less about specific technical procedures and more about finding ways to regulate and standardize growing project complexity, budgets, and the new cadre of software engineers who had until recently attracted little notice. New approaches to programming took center stage in the push to make software development outcomes more reliable. These strategies included the use of structured programming techniques, adding language features to promote reliability, measuring software performance (metrics), and using integrated development environments (IDEs).[11] Most of these ideas would make their way into personal computing, too, although it would take a decade or more for the engineering practices to take hold.

Computer programming is sometimes envisioned as an individual task, but by the late 1960s, commercial software was typically constructed in groups. For example, in mainframe computing environments like the IBM System/360, it was necessary to hire an army of analysts, technicians, and software developers to build and maintain non-trivial systems. Productivity gains associated with the "division of labor" principle simply did not work when dividing up the tasks of a large software project. The new approach had to involve *teaming*. Grace Hopper subtly introduced this concept when she created the world's first computer language compiler in 1952, known as the A-0 system, which translated symbolic codes into machine language. Hopper completed the first draft of her work and then immediately shared it with associates to see if they could make improvements, a strategy that she also followed during the creation of FLOW-MATIC, the predecessor to COBOL.

10. Ensmenger, *The Computer Boys Take Over*, 196–197.

11. Stuart Shapiro, "Splitting the difference: the historical necessity of synthesis in software engineering," *IEEE Annals of the History of Computing* 19, no. 1 (1997): 25–54.

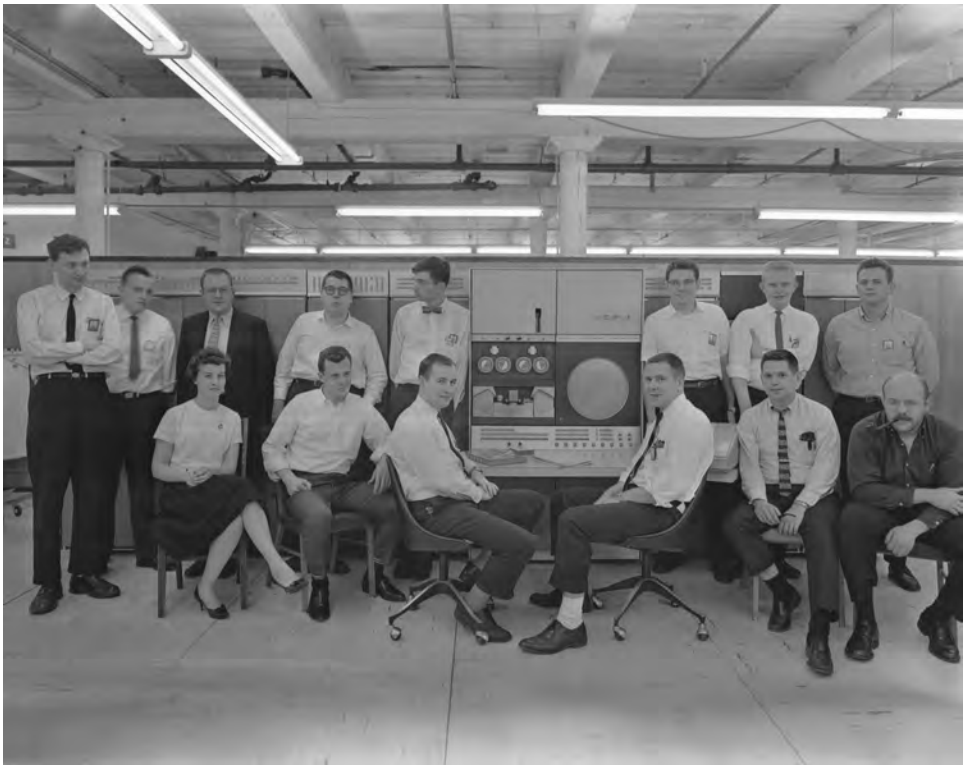


Figure 2.2 **By the 1960s, many engineering groups were using structure and clearly defined roles to bring predictability to their projects. The team that created the DEC PDP-6 computer was led by C. Gordon Bell, the man wearing a suit jacket in this 1964 photo.**

Standing L–R: Peter Sampson—Operating System Programmer, Leo Gossell—Diagnostic Software Programmer, C. Gordon Bell—PDP-6 System Designer (Creator), Alan Kotok—Operating System Lead, Russell Doane—Circuit Design Engineer, Bill Kellicker—Programmer, Bob Reed—Hardware Technician, George Vogelsang—Draftsman.

Sitting L–R: Lydia Lowe (McKalip)—Secretary to C. Gordon Bell, Bill Colburn—PDP-6 Project Engineer, Ken Senior—Field Service Technician, Ken Fitzgerald—Mechanical Engineer, Norman Hurst—Programmer, Harris Hyman, Operating Systems Programmer. (Courtesy the Computer History Museum and DEC)

2.2 The Complexity of Software

An underlying current of the 1968 NATO conference was that building software entailed a level of *complexity* that few fully recognized when digital electronic computers hit the scene in the 1950s.

But what made computer software so complex?

Let us start with a formal definition. Computer *software* is one part of a computer system that consists of data and computer instructions. Computer software is distinct from computer *hardware*, or the physical components of a computing system, such as the processors, circuits, boards, panels, monitors, switches, and other electronic devices in a machine.

A basic understanding of what software consists of changes over time, so it is useful to visualize a list of items that has taken shape in evolving contexts. Modern software includes a wealth of program *types* (operating system components, device drivers, application software, games, programming tools, malware), as well as supporting libraries, data, images, sound recordings, videos, email messages, Facebook posts, Tweets, and all manner of digital media. A *software release* typically consists of a bundled collection of items with a particular purpose, including a setup program, executable files, and hundreds (or thousands) of installed components, digital media files, documentation, and other resources.

From a business point of view, software may be considered a *commercial product* with economic value and utility, such as the popular applications GarageBand for iOS, Adobe Photoshop, or Microsoft Office. Software may also be distributed for free, such as open-source software or freeware. In these many contexts, one piece of software is distinct from another on a computer system, in the marketplace, or (under certain conditions) in copyright law. As recent historians have also discovered, each piece of software has its own history and impact, its creators and users. Software carries cultural memories and a society's hopes and fears.

In the 1960s, most software programs were supplied for free with the expensive computer systems that organizations purchased or rented from hardware manufacturers. In the U.S., IBM was the leading computer manufacturer by a large margin, followed by successful electronics firms like Burroughs, UNIVAC, NCR, Control Data, Honeywell, General Electric, and RCA. Corporations used mainframe computers for complex calculations, resource planning, bulk data processing, and transaction processing, including managing shipping, payroll, and employee records. In these many contexts, organizations needed to adapt the free software that they were given to match the needs of their customers. They needed to hire and train programmers and technicians to accomplish this work.

As computers grew and took on more tasks, the ailments plaguing software could often be traced back to one principle cause—*system complexity*. The complexity of software was engendered by programming's abstract nature and the scientific principle that a program constitutes a digital (discrete state) system based on

mathematical logic rather than an analog system based on continuous functions.[12] As software systems were being constructed with growing sophistication, project designers needed to consider numerous interrelated factors in their solutions, including the organization's list of requirements for a system (clearly stated or not), operating constraints related to hardware and software platforms, technical conditions within the computer itself (including memory resources), and the wide range of possible inputs and outputs that a program might encounter as it completed its work.

Real-world computer systems were designed so that they used only a prescribed set of resources, such as memory and processor time. From the point of view of the programmer, additional complexity arrived in the selection of programming languages, data structures, algorithms, flow control mechanisms, error handling structures, and the use of inherited source files and legacy code from other projects.

Individual computer programmers also brought their own tastes and psychological experiences to a project, as well as diverse training and educational experiences. All of these variables made the precise functionality of programs hard to predict, in the same way that storms and atmospheric conditions are challenging to forecast. The intricate balancing act was magnified in myriad ways as the responsibility for building new systems was distributed among team members who had different abilities and often coded in different locations and contexts.

In the late 1960s, anxious managers noted that the complexity of large systems created engineering problems with no easy solutions or mechanisms for assessment and control. As E.E. David of Bell Laboratories pointed out at the Garmisch conference,

> Production of large software has become a *scare item* for management. By reputation it is often an unprofitable morass, *costly and unending*. This reputation is perhaps deserved. No less a person than T. J. Watson [Jr., Chairman and CEO of IBM] said that OS/360 cost IBM over 50 million dollars a year during its preparation, and at least 5000 man-years' investment.[13]

In David's telling example, the software development project for OS/360 (IBM's operating system for the 360 series of computers) was famously late and over budget, problems blamed on poor management practices and unwieldy development teams. The project became the subject of Frederick Brooks's well-known guidebook on managing software projects, *The Mythical Man-Month* (1975), which we

12. Shapiro, "Software Engineering," 20.

13. Naur and Randell, *Software Engineering Report*, 13. The italic formatting is mine.

Figure 2.3 **A panel session from the 1968 Conference on Software Engineering in Garmisch, West Germany. Addressing the group is M. D. (Doug) McIlroy.** (Photo by Robert M. McClure and used with his permission)

will return to when I analyze commercial programming cultures and integrated software suites in Chapter 11.

2.3 Systems are for Customers

Who were these early software systems designed for, and what percentage of the population did they actually represent? Playing back conversations from the 1960s, it is sometimes hard to tell. Here is one fascinating transcript from the 1968 conference that we have been using as a touchstone. It involves six prominent leaders from the global computer industry, including two computer manufacturers, three academics, and one of the conference's organizers.[14] (See Figure 2.3.) The exchange focused on the *users* of computer software and the extent to which customers

14. For the original transcript, see Naur and Randell, *Software Engineering Report*, 24–25. The speakers were Professor J. N. P. Hume (University of Toronto), J. D. Babcock (Allen-Babcock Computing, New York, NY), Professor J. Berghuis (Philips' Computer Industrie [a Dutch computer manufacturer], Netherlands), J. W. Smith (Scientific Data Systems, El Segundo, CA), Dr. M. Paul (Leibniz-Rechenzentrum [a computing research center], Munich), and Professor A. J. Perlis (Carnegie Mellon, Pittsburgh, PA).

should be included in the design of new software —an emphasis which appears to be lacking in the first systems. A cautious professor J. N. P. Hume begins the dialog. (The *italic* is mine.)

> J. N. P. Hume [University of Toronto]: One must be careful to *avoid over-reacting to individual users*. It is impossible to keep all of the users happy. You must identify and then concentrate on the requirements common to a majority of users, *even if this means driving a few users with special requirements away*. Particularly in a university environment you take certain liberties with people's freedom in order to have the majority happy.
>
> J. D. Babcock [Allen-Babcock Computing, New York, NY]: In our experience the users are *very brilliant people, especially if they are your customers and depend on you for their livelihood*. We find that every design phase we go through we base strictly on the users' reactions to the previous system. *The users are the people who do our design, once we get started*.
>
> J. Berghuis [Dutch professor and industry consultant]: Users are interested in systems requirements and buy systems in that way. But that implies that they are able to say what they want. *Most of the users aren't able to*. One of the greatest difficulties will be out of our field as soon as the users realize what kind of problems they have.
>
> J. W. Smith [Scientific Data Systems, El Segundo, CA]: Many of the people who design software refer to users as 'they', 'them'. *They are some odd breed of cats living there in the outer world, knowing nothing, to whom nothing is owed*. Most of the designers of manufacturers' software are designing, I think, for their own benefit — *they are literally playing games*. They have no conception of validating their design before sending it out, or even evaluating the design in the light of potential use. The real problem is training the people to do the design. *Most designers of software are damn well incompetent*, one way or another.
>
> M. Paul [Leibniz-Rechenzentrum, Munich]: *The customer often does not know what he needs*, and is sometimes cut off from knowing what is or what might be available.
>
> Alan Perlis [Conference organizer, Carnegie Mellon University, Pittsburgh, PA]: Almost all users require *much less* from a large operating system than is provided.

Imagine a heated conversation among influential professors and experts who think that they each know best, and you've got the gist of this exchange. But notice how divided the industry leaders are about computer users and the design of software

in the 1960s. Each person is wondering: who are these systems really designed for? How much functionality should be provided to users?

Professor Hume outlined a realist position: You cannot make all of the software users happy, but if you try to satisfy a majority of them, you will probably find a good balance. Hume believed that it was legitimate to take away the freedoms of users on occasion to protect the system or satisfy the needs of the majority. We are still wrestling with the implications of this statement for security and civil liberties in electronic environments.

Bristling at the regimenting tone of this statement, James Babcock of Allen-Babcock Computing rushed to the defense of software users, whom he styled as "brilliant."

Allen-Babcock Computing was founded in 1964 in Los Angeles by James Babcock and Michael Jane Allen Babcock to profit from the rapidly expanding market for computer time-sharing services. Between 1965 and 1966, the company participated in the development of Conversational Programming System (CPS), a time-sharing product that ran under IBM's popular OS/360. In 1969, Allen-Babcock held a 3% share of the time-sharing services market—a lucrative position in this developing field.[15] Babcock's direct experience with "external" computer users like this is unusual but important to catch. He had regular contact with paying customers in a competitive marketplace outside of academic or corporate contexts, where "control" over the system typically trumped the preferences of users. In Babcock's opinion, if computer users do not directly enjoy the benefits of a new system, it is of little value.

But can computer users really be trusted? Professor Berghuis replied to Babcock that in his opinion customers had the right to make their selections, but most users do not know how to do it. If they only knew what computers were capable of, Berghuis mused, then software construction would be so much easier. Here, Berghuis puts the burden on users to *know* what they need, rather than on software designers to assess a customer's requirements.

J. W. Smith of Scientific Data Systems (SDS), El Segunda, California, supported James Babcock's position and he also spoke as an advocate for *users*.

SDS was a computer company established in 1961 by Max Palevsky and Robert Beck, veterans of the engineering firms Packard Bell and Bendix. This firm was an early adopter of integrated circuit technology and also a maker of time-sharing systems. (They sold their successful operation to Xerox in 1969.) Accordingly,

15. Harvard Business School, Lehman Brothers Collection, Allen-Babcock Computing Inc. https://www.library.hbs.edu/hc/lehman/Data-Resources/Companies-Deals/Allen-Babcock-Computing-Inc. Accessed August 20, 2019.

Mr. Smith also knew something about customers, and he quickly pointed out that most users were treated *disdainfully* by system designers, who seemingly devised software systems for themselves. Their overdesigned programs felt to him like "games"—i.e., eccentric, inward-facing diversions, representing the tricks of show-offs rather than any real attempt to satisfy the needs of users.

Our final comment comes from Professor Alan Perlis (1922–1990), who offered a new point—that software systems often contain *too much* functionality. This was another voice of concern about system complexity, which Perlis knew would translate into rising costs and overdesigned systems. Perlis would certainly have known. He was a savvy computer scientist and administrator who organized one of the first academic computer science programs in the U.S. He also believed that computer programming should be taught widely in schools, and we will see later what contributions he made to the learn-to-program movement. For starters, Perlis designed the Internal Translator (IT) programming language and co-designed Algorithmic Language (ALGOL) with a committee of 13 computer scientists. He advocated for the rights of both programmers *and* users throughout his distinguished career.

Here is the point. By the late 1960s, computers and software systems had radically changed—and so had users and customers. During this era, software was becoming highly complex, and it was rapidly incorporating support for advanced features such as multitasking and time-sharing. Software was also becoming "unbundled" from hardware sales, and as the process moved forward commercial programs took on the attributes of modern consumer products. These features included improved design concepts, compelling lists of features, market-based pricing, product reliability, and customer support from the software makers and third parties.

The decoupling of software services began in the mid-1960s and gained momentum when IBM announced in June 1969, that it would begin pricing software separately from hardware during the coming year.[16] In less than a decade, software sales was re-organized into discrete channels, including lines for original equipment manufacturers (OEMs), retail stores, mail order customers, foreign translation markets, custom licensing products, and more. By the 1980s, boxed and shrink-wrapped software packages with attractive designs became the norm for the emerging PC industry, and these goods were manufactured in facilities that were governed by the best practices of supply-chain management, warehousing, fulfillment services, and accounting. In short, PC software markets were built on the firm foundation

16. Luanne Johnson, "A view from the sixties: how the software industry began," in *From 0 to 1: An Authoritative History of Modern Computing*, eds. Atsushi Akera and Frederik Nebeker (Oxford: Oxford University Press, 2002), 101–109.

of mainframe and minicomputer sales and support. Contrary to popular opinion, the corporate software products industry remained very strong in the U.S. for decades.[17]

The software crisis of the late 1960s created an important mythology about computing in Europe and the U.S., because it called into question the reliability of software systems and the development processes connected to them. Worries about complexity would remain in the software industry for decades, resurfacing again in the era of GUIs and enterprise computing in the 1980s and 1990s. To address the issue, software managers introduced engineering principles and encouraged their employees to work in teams that were efficient, on time, and under budget—desirables that became an obsession for later programming contexts. In a subtler way, the software crisis also elevated a new group in the history of computing—*users*, who through market influence and creative action would change how software products were designed and used. Over time, these customers would help to make technology companies among the most valuable corporations on earth.

2.4 The Counterculture Movement

In the 1950s and 1960s, the centers of mainframe computing research in the U.S. were to be found in the headquarters of IBM in upstate New York and in the academic labs of nearby Cambridge, Massachusetts. By the late 1960s and early 1970s, however, a relatively compact region of California between San Jose and San Francisco became a crucible not only for political protests and a thriving counterculture but also a new set of computing paradigms that would deeply influence the technical world.[18]

17. Not until 1998, when Microsoft overtook it, did IBM cease to be the world's largest software supplier. See Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: The MIT Press, 2003), 174.

18. A number of excellent books have explored the myths and mythmaking of Silicon Valley entrepreneurs and software developers, including (by date of publication), Steven Levy, *Hackers: Heroes of the Computer Revolution* (Garden City, NY: Anchor Press/Doubleday, 1984; Revised edition, Sebastopol, CA: O'Reilly, 2010); Theodore Roszak, *From Satori to Silicon Valley: San Francisco and the American Counterculture* (San Francisco, CA: Don't Call It Frisco Press, 1986); John Markoff, *What the Dormouse Said: How the Sixties Counter-culture Shaped the Personal Computer Industry* (New York: Penguin Books, 2005); Fred Turner, *From Counterculture to Cyberculture: Stewart Brand, the Whole Earth Network, and the Rise of Digital Utopianism* (Chicago: University of Chicago Press, 2006); Michael Swaine and Paul Freiberger, *Fire in the Valley: The Birth and Death of the Personal Computer*, Third Edition (Dallas, TX: The Pragmatic Bookshelf, 2014); Walter Isaacson, *The Innovators: How a Group of Hackers, Geniuses, and Geeks Created the Digital Revolution* (New York: Simon & Schuster, 2014); Clive Thompson, *Coders: The Making of a New Tribe and the*

A seminal text in the communication of counterculture values was Theodore Roszak's *The Making of a Counter Culture* (1969), which criticized the dominant industrialized cultures of Europe and America and suggested new ideals for disaffected citizens, students, and intellectuals.[19] Roszak rejected what he called *technocracy* in modern societies, the oppressive regimes of corporate and technological expertise that seemingly dominated society and regimented social and intellectual life. His work echoed themes from other works of the period, including C. Wright Mills' *The Power Elite* (1956), Leo Marx's *The Machine in the Garden* (1964), Jacques Ellul's *The Technological Society* (1964), and Lewis Mumford's *The Myth of the Machine* (1967). Technology had its merits, these texts argued, but in the era of cold wars, nuclear weapons, and the expanding military-industrial complex, technology could also become a force of dehumanization. To reject this mindset required a transformation of consciousness, a mode of transcendence stimulated by new types of knowledge and collaborative styles of living.

If this social and political protest seems like a rejection of the mainframe computing culture that we have just surveyed, it was—at least in part. Countercultural intellectuals came to view most of the scientists who worked on government and military projects as Big Brother loving bureaucrats who were supporting the wrong team. The fact that many of the employees who worked on these projects also had concerns about the ethics of the military-industrial complex was beside the point, at least for a while.

In his description of the counterculture movement, Fred Turner has identified two groups that envisioned the transformation of consciousness as the essential task for healing American society in the 1960s. The first group withdrew from society and formed egalitarian communes in places like northern California, Colorado, and New Mexico. These communes could be in rural or urban areas, but they were unified in their rejection of middle-class, Cold War America and its presumed values. The second group focused on mind-expanding experiences including sexuality, psychedelic drugs, music, and alternative spiritualties. These countercultural experimenters often remained in society but developed a similar utopian outlook to those who choose to live in the communes.[20]

Remaking of the World (New York: Penguin Press, 2019); Margaret O'Mara, *The Code: Silicon Valley and the Remaking of America* (New York: Penguin Press, 2019).

19. Theodore Roszak, *The Making of a Counter Culture: Reflections on the Technocratic Society and Its Youthful Opposition* (Garden City, NY: Doubleday & Co., 1969).

20. Turner, *From Counterculture to Cyberculture*, 31–34.

Figure 2.4 **Spectators at the New Games in 1973 watch as Stewart Brand, a leader of the counterculture movement, lays out sticks for a group activity. Brand became fascinated with the collective use of small-scale tools.** (Photo by ©Ted Streshinsky/CORBIS/Corbis via Getty Images)

Collectively, Turner labels the two groups *New Communalists*, and he draws attention to their unique interests in small-scale tools and technologies. Unlike many in the New Left—the political activists who rejected computers along with bombs, weapons, and other symbols of the military-industrial complex—the New Communalists found a role for tools in their worldview, especially if the tools could be used to disentangle corporate America from the military and their perceived stranglehold on society.

Foremost among Bay Area New Communalists was Stewart Brand (1938–), a charismatic writer and publisher who became an unofficial spokesman for the counterculture movement in the 1960s and 1970s. (See Figure 2.4.) Brand's comprehensive publication, *The Whole Earth Catalog*, proposed to offer small-scale tools to those who would transform consciousness and society, either through self-sustaining communes or individual expressions of love, learning, and harmony. The

first *Whole Earth Catalog*, published in Menlo Park in 1968, outlined its mission through a short statement on the first page:

> The Whole Earth Catalog functions as an evaluation and access device. With it, the user should know better what is worth getting and where and how to do the getting.
> An item is listed in the CATALOG if it is deemed:
> 1. Useful as a tool,
> 2. Relevant to independent education,
> 3. High quality or low cost,
> 4. Not already common knowledge,
> 5. Easily available by mail.
>
> This information is continually revised according to the experience and suggestions of CATALOG users and staff.[21]

This preface, explaining the importance of "tools" and how they were selected, appears in each published edition. The compendium appeared every 3 months or so, changing its focus with the seasons. Between 1968 and 1972, almost two million copies of *Whole Earth Catalog* were sold, and each edition contained new essays, tools, and reviews. The central organizing categories in the catalog included "Understanding Whole Systems," "Land Use," "Shelter," "Industry," "Craft," "Community," "Nomadics," "Communications," and "Learning." Within each category there were listings of mail order products, book reviews, scientific texts, photographs, and short articles from contemporary figures such as Buckminster Fuller, Wendell Berry, Marshall McLuhan, and Timothy Leary. The catalog was essentially a utopian mail-order directory stocked with materials that would inspire hippies and communalists to raise their consciousness, live peacefully, and make the world a better place.

The *Whole Earth Catalog* became the bible of sorts for countercultural groups like the New Communalists. While paging through several volumes of the oversized catalog in preparation for *Code Nation*, I was struck by the optimism and excitement of the movement, which grew to attract over 750,000 people and more than 10,000 communes across the U.S.[22] In the space of an afternoon, a typical reader is able to

21. This statement appeared on the title page of each catalog, one page after the image of the planet earth from space. (See Figure 2.5.) For an excellent introduction to the *Whole Earth Catalog* and its structure, see Caroline Maniaque-Benton, ed., *Whole Earth Field Guide*, with Meredith Gaglio (Cambridge, MA: The MIT Press, 2016).

22. Turner, *From Counterculture to Cyberculture*, 32.

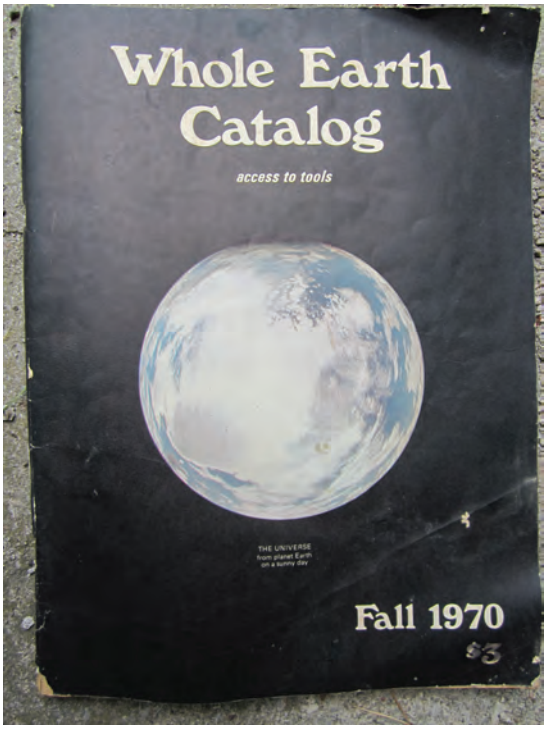


Figure 2.5 **Cover of the Whole Earth Catalog (Fall 1970).** (Courtesy of Getty Images, Glenn Smith, contributor)

learn something about growing crops, caring for farm animals, building basic structures, weaving cloth, generating power, preserving food, managing waste, providing for health care (including home births), keeping bees, building furniture, throwing pottery, establishing communal baths, meditating, and experimenting with mind-altering drugs. Providing essential tips for DIY communal living was the catalog's main purpose. All the book and merchandise reviews were positive, too. Discouraging product reviews were not printed as they supposedly transmitted "negative energy."

In terms of computing technology, the *Whole Earth Catalog* is surprisingly taciturn on electronics, computers, and software. Stewart Brand scholars tend to regard *The Last Whole Earth Catalog* (1971) as the definitive *Whole Earth* text because it offers the widest range of content, enjoyed the best sales, and was the winner of a national book award. Despite its comprehensive coverage, however, the 1971 *Last Whole Earth Catalog* features no computers, terminals, calculators, or software programs, nor do the editors write at any length about the burgeoning realms of

computer technology. Only in earlier volumes, such as the March 1969 *Supplement* to the catalog, can one find an occasional reference to computer-related technology, such as a photograph of a computer club meeting or a short advertisement for a calculator.

This might seem surprising, considering my earlier emphasis on the "crisis" mentality of the computer industry and the flurry of activity around the software engineering conference at Garmisch. But the reality was that the computer world was still in its infancy, a topic for government analysts and specialists in research labs. Most Americans had no direct experience with computers. At best, they caught glimpses of hulking mainframe units in films, television shows, and news broadcasts.

As in most things, Stewart Brand was a bit ahead of the curve. In December 1968, Brand had assisted Bay Area inventor Doug Engelbart at the so-called "Mother of All Demos" exhibition in San Francisco, demonstrating creative uses for computer terminals and the future of input devices and GUIs. But this moment aside, there were limits to what regular people knew about computers. Sophisticated electronics were especially rare in the rural communes or humble row houses that sheltered *Whole Earth Catalog* readers in the late 1960s and early 1970s. There was also a huge cultural gap between corporate computing and the average work experience of Americans. All of this explains why the only mention of computers in *The Last Whole Earth Catalog* is a textbook about how to create computer graphics from Prentice Hall, and a review of Nicholas Negroponte's new book on computer-aided design and human-computer interaction.[23] About the Negroponte book, Brand simply offered the quip in his publisher reviewer notes: "A book of beginning efforts to domesticate computers. Good intro to life with dumb-fuck genius machines."[24]

Brand and the editors of *Whole Earth Catalog* often included short notes like this about the products that they list. Negroponte's book was clearly innovative (he would go on to have an important career in computing), but Brand's note also passes along a common stereotype about electronic devices in this era—they appeared both *stupid* and *smart* at the same time. In other words, computers offered both *control* and *freedom* to users. Like the human soul, they required some measure of domestication and familiarity before transcendence might occur.

The *Whole Earth Catalog* was not an engineering manual. However, it spoke in metaphorical rhythms about small-scale tools that might elevate America's consciousness. The editors offered a compelling ideology: that low-cost instruments

23. Nicholas Negroponte, *The Architecture Machine* (Cambridge, MA: The MIT Press, 1970).

24. Stewart Brand et al., *The Last Whole Earth Catalog: Access to Tools* (San Francisco, CA: Portola Institute, 1971), 321.

that would soon return human communities to pre-industrial simplicity. Through the process, they would gradually raise group awareness. The counterculture movement had different priorities than high technology advocates, but the movement's *ideas* about tools were provocative and they left a lasting impression.

2.5 Everything is Deeply Intertwingled

The *Whole Earth Catalog's* busy, sprawling format was copied by numerous authors and innovators in the 1970s. The publication's homemade cut-and-paste quality, often utilizing different typefaces and texts rotated in opposing directions, became a recognizable standard for creativity, free thought, and challenging the *status quo*.

One of the most influential adapters of this style on the West Coast was Ted Nelson (1937–), author of the iconic idea and design book *Computer Lib/Dream Machines* (1974). It is not an overstatement to say that this book and its positive message about computers and computer literacy changed computing history.[25]

Ted Nelson was a prolific writer and inventor who studied philosophy and sociology, composed a rock-and-roll musical, made films, taught in the humanities, consulted in corporate and academic contexts, and completed distinctive work as an artist and designer. While Stewart Brand did not make many overt connections to computer technology in *Whole Earth Catalog*, Nelson oozed enthusiasm for computing in his writings, using his self-published *Computer Lib/Dream Machines* as a call to action for learning about computers and leveraging their power for good. As Nelson's ideas and enthusiasm attracted followers, he settled in the San Francisco Bay Area, making friends with engineers and inventors at the Home Brew Computer Club, Stanford Research Institute, and other high-tech hubs. Nelson would pursue a fascinating career in computing as a visionary and futurist, coining the term *hypertext* (text on a computer screen linked to webs of other texts), and envisioning a graphical, compound-document system that he called Xanadu. Hypertext and compound-documents would eventually be adopted in a variety of contexts in the computing world, including Tim Berners-Lee's implementation of the World Wide Web. I am grateful for Dr. Nelson's gift of an unpublished photograph from his collection for this book (see Figure 2.6).

Computer Lib/Dream Machines was published just before the introduction of PCs, but the revolutionary nature of "individualized computing" was clearly anticipated in Nelson's book. For this reason, it is helpful to define "personal computing"

25. For important assessments of Nelson's work, see Peter Morville, *Intertwingled: Information Changes Everything* (Ann Arbor, MI: Semantic Studios, 2014); and Douglas R. Dechow and Daniele C. Struppa, eds., *Intertwingled: The Work and Influence of Ted Nelson* (Heidelberg, Germany: Springer, 2015).

Figure 2.6 **Detail of Ted Nelson (reclining) with members of the Project Xanadu group, 1981.** (Photo from the collection of Ted Nelson and used with his permission)

as an interactive experience with computers that may include users on time-sharing systems as well as individuals using microcomputers or later personal computers.

Like *Whole Earth Catalog*, *Computer Lib/Dream Machines* is a compendium of information, organized in eclectic fashion around loosely-connected themes. The book's two titles are a reference to the way that the author presented his material—in overlapping, intertwining sections. In essence, there are two books bound together in one volume. You read *Computer Lib* to the book's mid-point or "pivot" page, and then you flip the book and read *Dream Machines* until you reach the end of *Computer Lib*. It is not necessary (or even useful) to read the book sequentially, however; the seemingly random organization of topics serves to emphasize the interconnected nature of information and its mystical *intertwingularity*, a term Nelson coined to express the complexity of interrelations among all forms of human knowledge.[26] For these theoretical ideas, Nelson is also recognized as a seminal figure in modern information theory and design.

26. On this term, Nelson wrote in *Dream Machines*: "EVERYTHING IS DEEPLY INTERTWINGLED. In an important sense there are no 'subjects' at all; there is only all knowledge, since the cross-connections among the myriad topics of this world simply cannot be divided up neatly."

As a document reflecting America's technology culture in the mid-1970s, *Computer Lib/Dream Machines* projects a positive, upbeat vision of computing society, but it also finds space for diatribes against IBM, U.S. intelligence agencies, the incompatibilities of computer systems, commercial television, "cybercrud" (computer jargon that serves to confuse), and the "ticking time bomb" of global population growth. (For more on the last issue, see Bob Albrecht's contemporary programming primer in Chapter 4.) Like the *Whole Earth Catalog*, Nelson adopts a countercultural point of view, but he offers technology as a way to improve the world, not abandon it. For Nelson, computers have fortuitously appeared as the next iteration in a long line of textual devices that have the potential to inform communities, expand the mind, and reunite people with their literary heritage. Interestingly, there is a strong liberal arts emphasis in his writing, evidenced through his deep appreciation for the classics of literature, art, history, sociology, psychology, biology, and mathematics.

Nelson came of age in the 1960s, and he knew firsthand about the "crisis" mythology surrounding corporate and government computing. He also recognized that the world of computers and software was changing rapidly. The 1974 edition of *Computer Lib/Dream Machines* envisioned a revolutionary computing context to be a terminal connected to a time-sharing system, providing interactive access to the mainframe's software and data resources. In the 1987 version of the book (see Figure 2.7), Nelson revised his presentation to introduce the wide range of computing technologies, including the Altair microcomputer, the Apple II, various IBM PCs and compatibles, the Macintosh, new minicomputer systems, and platforms running CP/M, MS-DOS, Unix, and Macintosh Finder. Regardless of the device, however, Nelson argued that computers only become revolutionary when the *user* was put in charge of the device and its resources. An important aspect of this command and control included computer programming. "The world is divided," Nelson intoned, "into people who have written a program and people who have not."[27]

Inspired by his influence, Stewart Brand described Ted Nelson as "the Tom Paine of the PC Revolution."[28] Nelson spread the message that corporate computing had become paternalistic and compartmentalized to the point that users had been removed from the decision-making process. As a result, computing in America had become "an atrocious tangle of excellent incompatible pieces, well-intentioned

Ted Nelson, *Computer Lib/Dream Machines*, Second Edition (Redmond, WA: Microsoft Press, 1987), DM 31 [1974 Edition, DM 45].

27. Nelson, *Computer Lib/Dream Machines*, Second Edition, 40.

28. Stewart Brand, "Foreword," in *Computer Lib/Dream Machines*, Second Edition, by Nelson, ii.

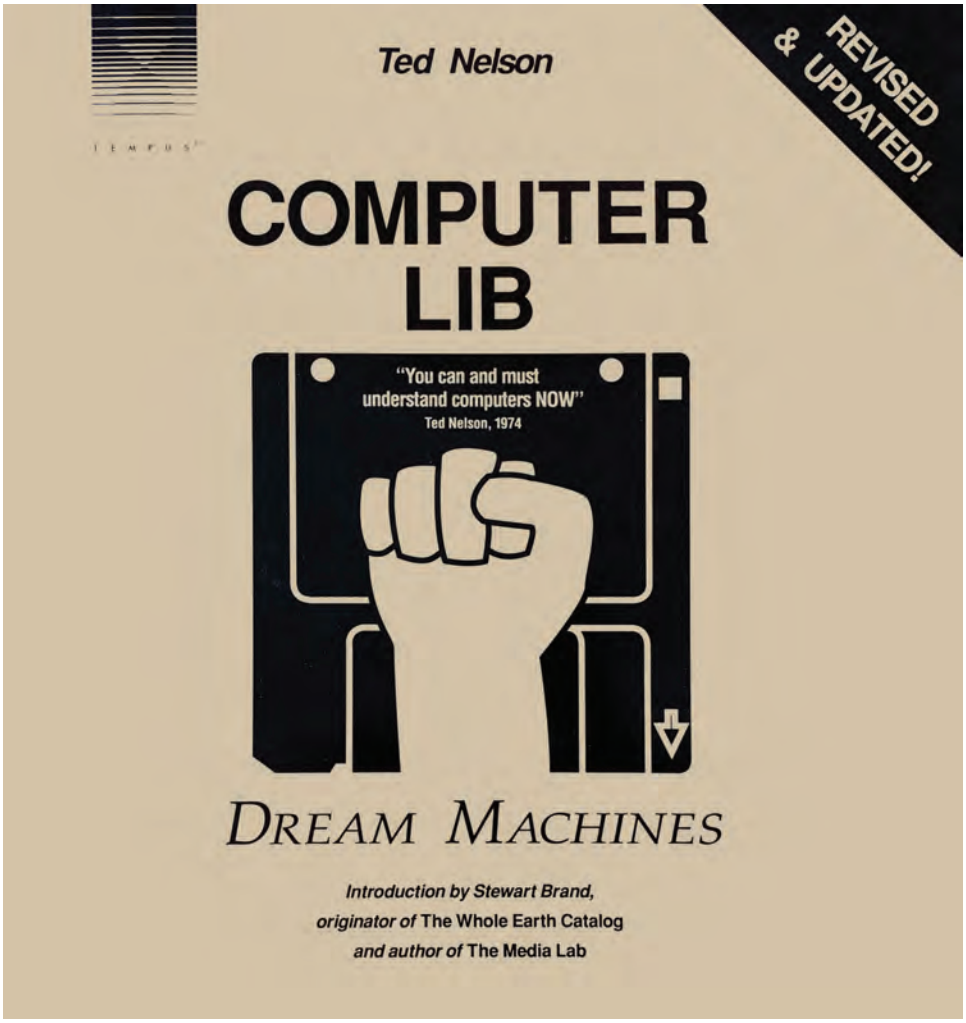


Figure 2.7 ***Computer Lib/Dream Machines*, 1987 Edition, by Ted Nelson.** (Used with permission from Microsoft)

incompatible junk, and inexcusable incompatible junk." Like many visionaries, Nelson pointed out both a crisis and a solution:

> We have to end this chaos. We have to re-unite the things that should never have been separate. We have to make it work for everybody. It is time indeed for *real* computer liberation.[29]

29. Nelson, *Computer Lib/Dream Machines*, Second Edition, 151.

2.6 The Birth of Computer Science

Ted Nelson emphasized liberating computer users and fixing software problems at an important moment in the history of computing. Although his ideas struck a different tone than many of the dominant paradigms in corporate computing, we should not try to separate his critiques too dramatically from earlier principles in the history of technology. This is an important caveat as we work to untangle the many strands of intellectual and cultural life that contributed to America's programming culture in the 1970s and 1980s. To understand why this is so, it is instructive to examine a few related developments that were taking place in the burgeoning discipline of computer science.

In the 1960s, as the Beatles invaded and IBM produced computers that put astronauts into orbit, the discipline of computer science emerged in universities across America. In a sense, this new academic field offered an institutional response to the radical changes that were taking place in the research labs of Cambridge, Massachusetts, and the San Francisco Bay Area. For those not steeped in academic teaching and administration (I have now worked as a college professor for 20 years), it is perhaps helpful to emphasize how daunting it can be to design and administer a new academic program—both for those who want to develop new fields of inquiry in a university setting, and for those outside the university who hope to benefit from a degree program and hire its graduates.

Most of the early digital electronic computers were not built by people that we would call "computer scientists." If the inventors who worked on computers were connected to universities at all, they came from the departments of Mathematics, Physics, Electrical Engineering, or Psychology. Often the computers were created by partnerships between academic institutions and government agencies. If financial support *did* come to the universities involved, they spent the money on new computing centers, research labs, student-faculty research teams, or other administrative projects. The first American universities to benefit from this type of support included Harvard, Princeton, Massachusetts Institute of Technology (MIT), Carnegie Mellon, the University of Pennsylvania, and Stanford. Despite the funding for computer-related research, however, actual courses in programming and computational logic were rare. When they did take place, they were distributed widely across the university. For example, a course in FORTRAN programming might be as likely in a physics department as in the mathematics department or the school of engineering.

During the 1950s, some momentum was established to begin a new academic discipline called "computer science" that might take up the theoretical analysis of computers and formal computing methods. In this context,

the world's first computer science degree program may have been the Cambridge Diploma in Computer Science, initiated at the University of Cambridge Mathematical Laboratory (UK) in 1953. This is the location where the pioneering EDSAC stored-program computer was first developed, and the new degree program was devoted to understanding and expanding this important system. (See Figure 3.5.) I will introduce the inventors of this computer and the first programming primer written for the system in Chapter 3.

In 1959, Louis Fein published an article in the newly-minted journal *Communications of the ACM* suggesting that computer science should be introduced as an academic discipline wherever universities could support it, including within the U.S.[30] According to Fein, the goals of this program should include fostering fundamental research in computing, educating graduate students, and preparing programmers for useful work in science, business, and industry. Universities were encouraged to build their programs by drawing expertise from across the institution, including the departments of Mathematics, Library Science, Economics, Business, Physics, and Engineering. Not long after this invitation, the first American Computer Science degree program was founded at Purdue University in 1962. In the following years, computing courses and new computer science programs were founded in several colleges and universities across the U.S.

Characteristic of the founding years of computer science programs was the leadership of administrators such as Professor Alan Perlis of Carnegie Institute of Technology (later Carnegie Mellon University). Perlis managed his institute's computing center, recruited faculty and graduate students, chaired the Department of Mathematics, and encouraged his colleagues to establish a new curriculum in computer science.[31] We first encountered Dr. Perlis in our discussion of the 1968 NATO conference, where Perlis served as a conference organizer and panelist. He is remembered as a seminal figure in computing not only for his publications, but for his effective leadership and skill in negotiating across academic boundaries, a task that he approached with political savvy and a sense of humor. After about 5 years of preparatory work, Carnegie Tech formally established a Ph.D. program in computer science in 1965, and Perlis became its principal director.

Professional organizations were also useful in the founding of university computer science programs, and none more so than the ACM. In 1968, the ACM

30. Louis Fein, "The role of the university in computers, data processing, and related fields," *Communications of the ACM* 2, no. 9 (1959): 7–14.

31. The founding of Carnegie Mellon's Computer Science department is described in an important interview with Alan Newell, a colleague of Perlis'. See Allen Newell, "An interview with Allen Newell," interview by Arthur L. Norberg, Charles Babbage Institute (University of Minnesota), June 10–12, 1991, 34.

proposed a formal curriculum for computer science in a report that recommended an appropriate selection of thematic courses. The organization advertised their plan with a series of articles about major concepts and theoretical concepts in the new field.[32] Over the next decade, the ACM and its standing committee, the Curriculum Committee on Computer Science, hosted conferences that helped smaller universities adapt the initial recommendations for their specific contexts. Although smaller schools might not have graduate students or large budgets, the interdisciplinary nature of computer science adapted well to the distributed curriculum of many liberal arts colleges, especially those with strengths in writing, mathematics, art, history, music, and philosophy. (Several of the PC industry's early leaders studied computing in these small college contexts, such as Ted Nelson at Swarthmore College, Cynthia Solomon at Radcliffe College, and Steve Jobs and Peter Norton at Reed College.)

During the 1960s and 1970s, the computer science curriculum accentuated mathematics, including courses in calculus, Boolean algebra, numerical analysis, and statistics. The courses that introduced programming taught fundamental concepts in high-level languages like FORTRAN, ALGOL, Pascal, and C. In later courses, students received an introduction to data structures and algorithms. (COBOL and BASIC were offered less often, and remained controversial because of their perceived lack of structure, an issue that will be explored in Chapter 5.) Assembly language was often introduced as an aspect of computer architecture. In the years to come, computer science departments expanded their offerings to include courses in operating systems, artificial intelligence, automata, and theories of computation.[33]

Representative of the emerging computer science faculty is Edsger Dijkstra (1930–2002), a Dutch scientist who started his undergraduate career as a student of theoretical physics but switched fields and completed a doctorate in 1959 on assembly language programming strategies for an early Dutch computer.

Dijkstra entered computer science during a time of rapid growth in computer architecture. Riding a surge of interest in computing in the Netherlands, Dijkstra worked as a programmer at the Mathematisch Centrum in Amsterdam from 1952 to 1962. In 1962, Dijkstra accepted an offer to become a professor of Mathematics at the Eindhoven University of Technology, where he developed theoretical concepts in computer science and supervised numerous graduate students. (Many of these students became leading figures in the field in the following years.) Soon Dijkstra weighed in on the looming "crisis" in software reliability

32. Ensmenger, *The Computer Boys Take Over*, 133.

33. Ensmenger, *The Computer Boys Take Over*, 129.

and complexity, and he attended the 1968 engineering conference in Garmisch. To address his concerns, Dijkstra introduced new engineering practices, devised seminal algorithms, and he emphasized the importance of applied mathematics in the new field—all contributions that influenced the subsequent development of computer science. He also advocated for structured programming concepts and famously attacked the BASIC programming language, publishing a negative article about the language's lack of structure and the dangers of its "GOTO" keyword. (See Chapter 5.)

In 1984, Dijkstra transferred to the growing Computer Science department at the University of Texas at Austin, where he continued to direct research until his retirement in 1999. Through numerous graduate students and colleagues, Dijkstra guided seminal work in language research, compiler design, concurrent programming, modeling, and operating systems. Throughout his career, the Dutch scientist was a theoretical purist, believing that individual implementations of computing technology were less important than the formal abstract knowledge gained through research.[34]

Computer science emerged as a new discipline in the natural sciences, and it established a place in the university curriculum. However, few courses were introduced at the high school level, a problem we will return to later in this book. Still, between the early 1960s and the early 1980s, new computer science programs appeared in American colleges and universities at a rate of about 20% annually.[35] At the height of the major's popularity, more than 5% of all U.S. male college students would graduate with a degree in Computer Science or Information Science. Moreover, in 1984, approximately 37% of the U.S. Computer Science graduates were women, indicating widening opportunities in the field, strong demand, and the program's early success in reaching at least some underrepresented groups.[36] If the goal was bolstering the pool of potential software developers for industry, the project seemed like a success.

So how did the U.S. computer industry respond to the new field and its curriculum? Although many employers were positive, there also those who critiqued the way that academics continually gravitated to theoretical principles over practical applications. More a tendency than a hard and fast rule (and therefore fertile ground for myth-making), this inclination was most apparent in corporate contexts where large teams of IT programmers were assigned to build information processing systems. At the time, many working programmers still had no formal education

34. Tedre, *The Science of Computing*, 6.

35. Ensmenger, *The Computer Boys Take Over*, 115.

36. Abbate, *Recoding Gender*, 3.

in computing. They learned on the job through programming primers and other printed resources. As a result, these developers were learning to code in relatively narrow contexts.

In the 1960s and 1970s, the conflict between academic computer scientists and professional programmers became severe in some circles, driving a wedge between the two subcultures that remains discernable up to the present. As Nathan Ensmenger has commented, "Computer scientists expressed disdain for professional programmers, and professional programmers responded by accusing computer science of being overly abstract or irrelevant."[37]

Partisans in the dispute found ways to deploy stereotypes to heighten the impact of their critiques. In 1968, Hal Sackman was a research associate working for IBM, studying the characteristics of programmers employed in the commercial computing industry. In a report published that year, Sackman wrote: "They [computer science professors] are too busy teaching simon-pure courses in their struggle for academic recognition to pay serious time and attention to the applied work necessary to educate programmers and systems analysts for the real world."[38] By *simon-pure*, Sackman implied that academics were engaged in teaching computing concepts with an abstract, theoretical propriety. Sackman believed this to be a pretentious commitment to principle that was essentially insincere and disconnected from everyday concerns.

Richard Hamming, a mathematician, information theorist, and Turing Award winner at Bell Labs, also criticized the academic training delivered in his era. In 1968, Hamming repeated a phrase that we have already encountered in the Garmisch conference transcripts:

> At present there is a flavor of "game-playing" about many courses in computer science. I hear repeatedly from friends who want to hire good software people that they have found the specialist in computer science is someone they do not want. Their experience is that graduates in our programs seem to be mainly interested in playing games, making fancy programs that really do not work, writing trick programs, etc.[39]

Recall that J. W. Smith of SDS also made the same accusation against elite programmers whom he saw as "playing games" in their solutions. This is an attack akin to the mythical "ivory tower" trope of academic research, which supposedly rises in isolation above everyday concerns.

37. Ensmenger, *The Computer Boys Take Over*, 129.

38. Ensmenger, *The Computer Boys Take Over*, 133–134.

39. Ensmenger, *The Computer Boys Take Over*, 134.

As a partial reply, several academics offered the counter charge that computer science was being unfairly equated with simply teaching programming skills. In fact, they argued, writing computer programs is just one of the proficiencies that Computer Science students need to learn. ACM Past-President Peter J. Denning emphasized this point in his discussion of the attributes of a thorough Computer Science education:

> Every practitioner of the discipline [computer science] must be skilled in four basic areas: algorithmic thinking, representation, programming, and design... Even though everyone in the discipline is expected to know these skills, it is a mistake to equate computer science with any one of them, e.g., programming... There are many aspects of the discipline that do not involve programming even though they involve algorithmic thinking, representation, and design.[40]

This disjuncture between the academy and industry remains an important dynamic of modern computing culture, and it has engendered several myths, with each side accusing the other of infringements and narrow thinking. The fissure has also influenced the learn-to-program movement, because self-taught programmers only seek out some of the skills that they need, and they are often forced to chart their own course through mounds of textbooks, software manuals, programming forums, and boot camps without adequate mentoring or support. On the other hand, many self-taught programmers have done very well without academic support and degree programs, and they feel that the essence of computing relates to learning by doing, forming their own communities, and participating in the dynamic worlds of business and commerce—not learning to climb the ivory tower. Personal computing has been highly influenced by these debates.

2.7 Computers for the People

San Francisco was a major hub of computing activity in the 1970s, both before and after the so-called "PC Revolution." In the years before the introduction of the first microcomputers, the Bay Area was replete with high tech startups, engineers, hippies, intellectuals, and students advocating for change. A fascinating representation of these overlapping mindsets can be found in the life and experiences of Lee Felsenstein (1945–), a Bay Area antiwar protestor with a Berkeley Electrical Engineering degree who combined political activism with a unique vision for computing technology. Felsenstein was one of the original members of the Homebrew

40. Peter J. Denning, "Computer Science: The discipline," in *Encyclopedia of Computer Science*, Fourth Edition, eds. A. Ralston, E. Reilly, and D. Hemmendinger (Hoboken, NJ: Wiley, 2003).

Computer Club, the influential group of tinkerers and entrepreneurs who began meeting about electronics and computing in Menlo Park in 1975. The club's first meeting allowed participants to preview one of the early MITS Altair microcomputers, the famous device based on the Intel 8080 microprocessor that was revealed to popular acclaim in the January 1975 issue of *Popular Electronics*. At first there were 32 members active in the club. After about 6 months, the group had expanded to about 100 regular attendees, with a newsletter distribution of almost 300.[41]

At Homebrew, Felsenstein met several people who would make major contributions to the development of PCs, including Gordon French, Fred Moore, Adam Osborn, Steve Wozniak, and Steve Jobs. Through these contacts, Felsenstein also came to know Stewart Brand, Ted Nelson, Bob Albrecht, and other entrepreneurs and authors interested in print publishing.

Felsenstein took part in the Free Speech Movement in Berkeley in 1964–1965, the first massive act of civil disobedience on an American college campus in the 1960s. The events at Berkeley were influenced by New Left politics, and they were deeply connected to the Civil Rights Movement and Vietnam War protests. Many students took part in all three struggles and saw them as part of the same cause. Figure 2.8 shows one creative way that Felsenstein used his engineering skills to prepare for these events. In 1969, he designed a device he called a "decentralized bull horn" (FR-3), which allowed protesters to communicate in crowded situations typical of student movements.[42] (See Figure 2.8.) The bull horn was designed to have input and output connectors so that many individual devices could be driven by one lead device. Using these tools, a crowd of protesters armed with the bull horns could speak to each other when crowd noise became too loud for regular communication. It was an engineering solution designed to help people participate effectively in free speech rallies without the loudness and distortion of contemporary systems.

Felsenstein's contributions to the movement usually brought together his love of technology with computing solutions that would help regular people. An early inspiration for his work was Stewart Brand's *Whole Earth Catalog*, which connected Americans through print to a positive message on a massive scale. More theoretically, Felsenstein was also influenced by Ivan Illich's book *Tools for Conviviality* (1973), which offered a conceptual rationale for using "limited" tools that might improve the lives of average citizens. Ivan Illich (1926–2002) was a Roman

41. Elizabeth Petrick, "Imagining the personal computer: conceptualizations of the Homebrew Computer Club 1975–1977," *IEEE Annals of the History of Computing* 39, no. 4 (2017): 27–39, here at 31.

42. I thank Lee Felsenstein for his detailed description of the decentralized bull horn system, which he explained via email correspondence in June 2019. Felsenstein also granted me permission to reproduce the image in Figure 2.8.

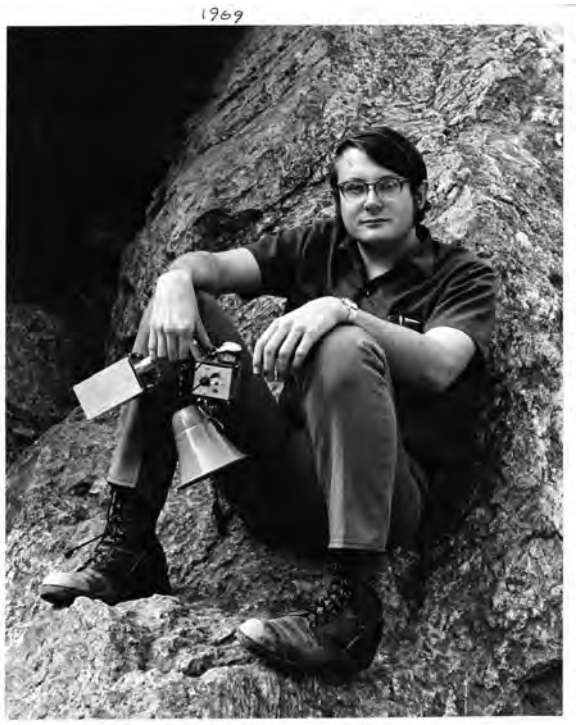


Figure 2.8 **Photo of Lee Felsenstein with a decentralized bullhorn that he designed in 1969.** (Courtesy of the Computer History Museum, used with permission of Lee Felsenstein)

Catholic priest and a staunch critic of mainstream, institutionalized education; he argued that a society functioned well when it made education broadly available and resisted the formation of elite groups that controlled and monetized the flow of information. Ideally, Illich wrote, learning would be hands-on, low-tech, and socially beneficial—what he defined as "convivial."[43]

Felsenstein was an electrical engineer by training and he had worked with computing systems at UC Berkeley and the Ampex Corporation, the later a maker of multitrack tape recording devices. Inspired by Illich and the Free Speech Movement, Felsenstein recognized in transistor technology the potential to offer citizens inexpensive access to communication tools and abundant sources of information. Like Stewart Brand and Ted Nelson, Felsenstein believed that appropriate tools could elevate the consciousness of average citizens and promote social change.

43. "Total learning expands when the range of spontaneous learning widens along with access to an increasing number of taught skills and both liberty and discipline flower." Ivan Illich, *Tools for Conviviality* (New York: Perennial Library, Harper & Row, 1973), 59.

"We were looking for nonviolent weapons," Felsenstein wrote, "and I suddenly realized that the greatest nonviolent weapon of all was information flow."[44]

In August 1973, Felsenstein and four others created The Community Memory project in Berkeley, California. The goal of this endeavor was to build a simple time-sharing computer system that could function as a hub for information and community organizing in the region. The first Community Memory system was established in Leopold's Records in Berkeley, a popular hangout for students, musicians, poets, and counterculture types of all ages. (For more about this organization and its mission, see Chapter 7.)

The Community Memory information hub consisted of a teleprinter and a keyboard, surrounded by a simple cardboard case to protect the system and reduce the shrill noise that emitted from the teleprinter. The device was connected via a 110-baud link to a reconditioned Scientific Data Systems 940 time-sharing computer in San Francisco. By using the teleprinter and simple commands, novice users could compose short messages, associate them with keywords, and post them to the system. Users could also search for messages or general topics of interest using keywords. There were helpful signs for users explaining how to operate the device, and typically a Community Memory volunteer was nearby to offer help and encouragement. The device functioned as a community bulletin board, where locals could get information about music, art, food, carpooling, protesting, and other activities. Users were not required to register or share their names to use the system. For many Berkeley residents, it was their first opportunity to see or use a computer.

As Community Memory gained momentum, Felsenstein developed a related project that he designed in the fall of 1974, the Tom Swift Terminal. Steven Levy introduced Felsenstein's project to the general public in his book *Hackers: Heroes of the Computer Revolution*.[45] For historians interested in the plan, the best source is Felsenstein's short vision document, which provides a high-level explanation of the circuitry and the components required to build the system.[46] I am fascinated with Felsenstein's hand-written prototype, because I see it as a forerunner of the learn-to-program movement, with its socially-based call to action and step-by-step instructions that taught computational thinking. Through the specification, Felsenstein and his colleagues argued for the democratization of computers; they made an appeal for *ordinary citizens* to program and use computers. It was a fascinating echo of the emphasis that John Kemeny and Thomas Kurtz had put on

44. Unpublished Felsenstein memoir, quoted in Isaacson, *The Innovators*, 299–300.

45. Levy, *Hackers*, 2015.

46. Lee Felsenstein, "The Tom Swift Terminal, or, A Convivial Cybernetic Device." http://www.leefelsenstein.com/. Accessed August 19, 2019.

bringing "computing to the people" during the development of time-sharing BASIC at Dartmouth College in 1963–1964.[47]

The Tom Swift Terminal was designed to be sturdier and easier to use than the Community Memory system, which was essentially a cardboard prototype showing how a community-centered technology might develop. The "Tom Swift" was named to honor America's "everyman" from literature who was fond of tinkering and experimenting far from the centers of corporate and government power. The terminal consisted of a box containing a bus, a power supply, and connections for printed circuit boards. When the owner hooked up a keyboard, a modem, and a television set, he or she had their very own functioning computer. (The schematic also allowed for a dialup modem connection that could connect to a time-sharing system.) After the connection was made, interactive "personal computing" could be accomplished through the remote computer's operating system and software. The user would experience the terminal session as lines scrolling on their home television set.

Lee Felsenstein wrote up the following goals for the system:

(a) to provide an inexpensive computer terminal useable in public-access information systems which is;

1. capable of using the home TV set as a character display. With hard copy as an add-on option.
2. easily useable by untrained people in a non-professional environment.
3. readily expandable by field modifications to higher levels of "intelligence" and off-line readability.

In an advertisement to publicize the concept, Felsenstein offered interested parties a 25-page booklet describing the proposed device for 50 cents. The advertisement made specific reference to Illich's *Tools for Conviviality*, describing a new approach to computing that would be non-industrial, fun, and playful. People would learn, understand, and repair this tool with little formal training, just like the tools humans used before the advent of industrial systems.[48]

2.8 Personal Computing

In the end, the Tom Swift Terminal would not become a commercial product. A few months after it was proposed, the Altair 8800 microcomputer kit was released

47. On this point, see Joy Lisi Rankin, *A People's History of Computing in the United States* (Cambridge, MA: Harvard University Press, 2018), 27. I introduce BASIC programming concepts in Chapters 4 and 5.

48. Lee Felsenstein, "Tom Swift Lives," printed by People's Computer Company, Menlo Park, California. http://www.leefelsenstein.com/ Accessed August 20, 2019.

by MITS in Albuquerque, New Mexico. Lee Felsenstein, Gordon French, and Fred Moore organized the first meeting of the Homebrew Computer Club to examine the device, and they attracted a cross-section of electronics enthusiasts from the region to discuss the Altair and other projects.

The next part of the story is better known, and the subject of popular books, television programs, and films. After the Altair, the microcomputing era took shape at a fast pace. In mid-1975, Bob Marsh, Lee Felsenstein, and Gordon French designed a new microcomputer around the Intel 8080 microprocessor called the Sol-20. The Sol was ready for commercial sale in December 1976. Their device created major excitement—it appeared as if the potential for low-cost computing was finally being realized.

Steve Wozniak and Steve Jobs also demonstrated a new microcomputer at the Homebrew Computer Club in July 1976—a homemade prototype later known as the Apple I. (See Figure 2.9.) About a year later, the Apple II microcomputer debuted. This machine was a more mature product with a custom plastic case, a printed circuit board, and slick modular components. Moving beyond the Silicon Valley circle of hobbyists, the Apple II became a catalyst for personal computing hardware and the nascent PC software industry across the country. In a cultural sense, the Apple II also reflected the aspirations and designs of Lee Felsenstein, Stewart Brand, Ted Nelson, and other prominent voices from the counterculture movement. Steve Wozniak summed up the connection to Ivan Illich's work in a special *Byte* article to commemorate the launch, "To me, a personal computer should be small, reliable, convenient to use and inexpensive."[49] Notice that Wozniak, too, used the emerging term, *personal computer*. In this early phase of personal computing, a PC was defined as a small, multi-purpose device that was relatively inexpensive to purchase (compared to minicomputers and workstations), and it was designed to be used by individuals.

In rapid succession, there came a series of PCs from different manufacturers: the Apple II (June 1977), the Tandy TRS-80 (August 1977), and the Commodore PET 2001 (October 1977). These three devices definitively launched what pundits later called the "PC Revolution," a new social order that fulfilled the promise of earlier microcomputer experiments with mass market products and opportunities.

What began as a push to provide simple computing tools to ordinary people culminated in a new sector of the computer industry. Behind the devices was a mythological belief that the tools would enrich the experience of average Americans, elevate their consciousness, and promote political change. This "revolution" produced unexpected results when the world's largest computer manufacturer

49. Steve Wozniak, "System Description: The Apple-II," *Byte* 2, no. 5 (May 1977), 34–43; here at 34.

Figure 2.9 **Steve Wozniak (left) and Steve Jobs with the Apple I Computer (1976). A repeating message on the displays reads, "Computer... available at BYTE Shop."** (Photo: Joe Melena. Image courtesy of the Computer History Museum and used with permission of Apple Computer)

chose to enter the nascent sector as well, releasing the IBM Personal Computer in August of 1981. This device was intentionally assembled from off-the-shelf products and did not have its own operating system, software, or programming tools.[50] However, IBM PCs and compatibles would soon carve out a lucrative niche in the PC industry as well. It took years for PCs to compete with mainframe and minicomputers in terms of revenue, but the PC industry had launched and soon there was a need for PC programmers and learning tools for both developers, business users, and hobbyists. Between 1981 and 1983, the receipts of PC software publishers grew from $70 million to $486 million.[51]

The stage was set for the rapid democratization of programming culture in the U.S. Computer literacy programs developed in the wake of widespread exposure

50. For more on the origins of the first IBM PC, see James W. Cortado, *IBM: The Rise and Fall and Reinvention of a Global Icon* (Cambridge, MA: The MIT Press, 2019), 379–418. I provide a more detailed analysis of this platform and its early software in Chapters 5, 6, and 9.

51. Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog*, 210.

to computers, as new users sat in front of PCs and wondered what to do with them. Computer programming, once considered the domain of corporate specialists, became a popular way to learn about computers and benefit from them. In the next chapter, we will learn more about this new skill, and how learning to program gained momentum as a popular movement in America.

3 FORTRAN, Logo, and the Tower of Babel

"The emergence of motion pictures as a new art form went hand in hand with the emergence of a new subculture... people whose skills, sensitivities, and philosophies of life were unlike anything that had existed before... Similarly, a new world of personal computing is about to come into being, and its history will be inseparable from the story of the people who will make it."

Seymour Papert, *Mindstorms: Children, Computers, and Powerful Ideas* (1980)

With the rapid introduction of personal computers (PCs) in the 1980s, computer programming grew from a relatively narrow technical and professional activity to a topic of fascination for a broad swath of the American public. Scientists, business people, educators, entrepreneurs, hobbyists, artists, and students—all were drawn by the pull of computing.

As MIT professor Seymour Papert argued in his popular book *Mindstorms*, PCs helped to create a new world filled with people who were being transformed by their technical experiences. Among the most positive interactions were the acquisition of new cognitive capacities, benefits that would come especially to those who learned to write their own programs. Papert claimed that the requisite stages of the programming process (problem solving, designing, coding, and debugging) all influenced the developing mind and a person's underlying cognitive abilities. This was especially true for the young, who with proper guidance could use programming computers as a way of jump-starting mental development, such as nurturing spatial and logical reasoning skills.

Papert was in a good place to develop his ideas and spread their influence; he joined MIT in 1963 during an intensive period of government-sponsored computer research that produced breakthroughs in time-sharing systems, operating systems, programming languages, artificial intelligence, and cognitive psychology. He later co-founded MIT's artificial intelligence lab with Marvin Minsky and cultivated a network of colleagues around the globe that spread his educational initiatives.

This chapter examines the origins of the learn-to-program movement in the U.S., an educational agenda that sought to introduce computational thinking to all people and gradually expand access to computers and programming-related skills. It explores the creation of the first programming languages, and early attempts to teach assembly language, FORTRAN, COBOL, and BASIC. I'll introduce the educational reforms proposed by Seymour Papert, Cynthia Solomon, and Wally Feurzeig at MIT, and the experience of elementary and middle school students who learned Logo under their influence. I will also highlight some of the author-programmers who devised the first programming primers, including Maurice Wilkes, Grace Mitchell, Donald Knuth, Daniel McCracken, and Daniel Watt.

The mythical "PC Revolution" put millions of computers in the hands of Americans. But for several years there was little in the way of commercial software for the new machines. In a very real sense, the early adopters of PCs needed to learn to program because they had little choice if they wanted to use their computers. But which programming language should they choose? Which PC platform should they create applications for? What steps should they follow to learn programming fundamentals, and how long might it take for a novice to become reasonably proficient as a software developer?

By the 1970s, many universities were offering Computer Science degrees that included programming instruction as part of the curriculum. But should American K-12 schools also introduce software development concepts? Would an aptitude for programming enhance core instruction in reading, writing, and mathematics? Or would programming simply overcrowd the curriculum, taking away valuable resources from spoken languages, art, science, music, and physical education? Finally, would learning to program *really* help most people later in life? Or were programming careers just rewards for a narrow slice of the population? In social and economic terms, could one *prove* that computational thinking was valuable to study?

The learn-to-program movement posed all of these questions, bringing forth a torrent of ideas, proposals, challenges, and products. Although the movement did succeed in introducing a generation to programming fundamentals, it also led to a splintering of educational resources and growing factionalism about which computer languages and platforms were the best. The educational movement's gradual decline in the mid-1980s had real consequences. It contributed to a decade or more of ambivalence about computer literacy, which devalued America's technical infrastructure and narrowed the understanding of who should consider computer-related occupations. As the movement continued, it traveled more corporate and commercial paths, becoming a manifestation of the nation's largest software companies. Today, pundits in the software industry still cannot agree on the best vocational

path for creating new programmers, a problem that has vexed organizations like the Association for Computing Machinery (ACM) since the 1960s.

We shouldn't be surprised with these mixed outcomes. The learn-to-program movement fits the pattern of many social and educational movements in American history, especially those that advocate for collective action and deep social change. The learn-to-program movement was inspired by charismatic leaders and economic necessity, and it gained momentum due to social, cultural, and economic factors. The movement benefited from waves of enthusiasm connected to the distribution of new PCs, and it also captivated talented writers and entrepreneurs who sought to teach programming fundamentals to average Americans. For a time, the movement seemed to unify the aspirations of many who sought to saturate the nation with computers, creating a shared reading of mythologies about computing and a vision to realize them. And then, in the mid-1980s, the educational underpinnings of the movement began to fracture, dissipating the cause's energy and impact, until a new wave took shape in more recent times.

3.1 Solving Problems with Computers

Computer programming is a catch-all term for problem solving with a computer. The core task of programming is to create a sequence of instructions in a computer language that will automate a given task or find a solution to a problem of interest.

Although early computer programmers devised their solutions by directly manipulating or "setting up" the wiring and circuitry of computers, by the late 1950s most programmers used computer languages to automate problem solving. Low-level languages (such as machine language and assembly language) provide instructions that are closely related to a computer's underlying architecture. High-level languages (such as FORTRAN and Java) provide instructions that are optimized for specific problem-solving requirements. Programs written in a high-level language usually have an additional advantage—they are more easily moved from one computing platform to the next.

In a modern context, writing a program usually entails the use of numerous software development tools, each with a specific purpose. The activity often takes place in a comprehensive integrated development environment (IDE), where the programmer can design the user interface, enter program code, adjust settings, review documentation, test and debug the application, and interact with online support communities. Present-day examples of an IDE include Microsoft Visual Studio for Windows, Xcode for Apple platforms, and Eclipse for Java development.

In the early years of computing, however, the programming tools for software developers were much more limited. A typical programmer would write out his or her instructions for the computer by hand, and then prepare them for entry into the

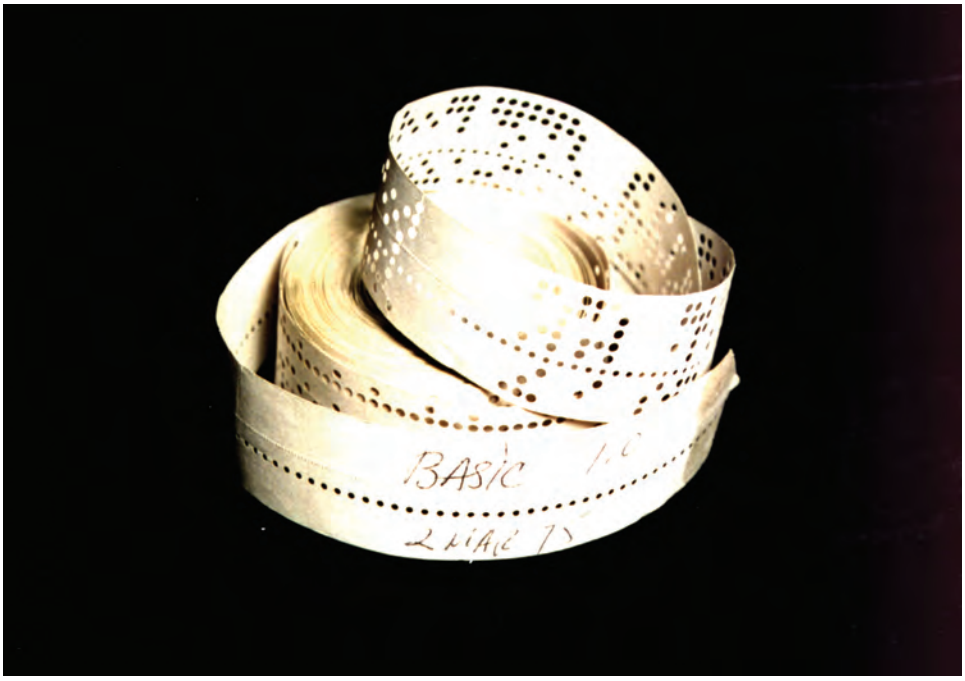


Figure 3.1 **Photograph of the punched paper tape for MITS ALTAIR BASIC 1.0, created by Bill Gates, Paul Allen, and Monte Davidoff for the Altair 8800 microcomputer. Dated March 2, 1975.** (Courtesy of the Computer History Museum)

computer's memory using input media such as punched cards, punched tape, magnetic tape, or (in later years) keyboard input. In these contexts, programming was a deeply *mental exercise* that only involved a computer in the later stages. For this reason, a fundamental step in the programming process was planning and research. Engineers worked to solve a computational problem as efficiently as possible and a major concern was always maximizing limited computer resources. Programmers prepared a program for the computer in its near-final form, and only later loaded the routines into memory. Fixing problems that arose often involved a painful process of trial and error.

A concrete example of this problem-solving approach comes from one of the first commercial programming languages written for microcomputers, the original BASIC interpreter created for the MITS Altair 8800 (See Figure 3.1). This program was written by Bill Gates, Paul Allen, and Monte Davidoff while Gates and Davidoff were students at Harvard.

When the Altair was announced in early 1975, there was no commercial software available for the machine. But computing enthusiasts soon realized that if

someone could create a BASIC interpreter for the Altair, then hobbyists could write their own BASIC programs on the computer and get the device to perform non-trivial work. (Chapter 4 explains why BASIC was chosen for this duty, despite its limitations.) Gates and his friends researched the Altair's specifications, and then they bought a book about the Intel 8080 microprocessor—the electronic "brain" of the Altair. As luck would have it, they had been working for some time with time-sharing systems, and they also knew how to write assembly language programs that could make efficient use of a computer's internal architecture.

To prepare for this project, Gates studied recent versions of the BASIC language (originally created by John Kemeny and Thomas Kurtz) and he scrutinized how the language operated. Then he designed an interpreter program that would provide Altair users with essential BASIC features while consuming as little computer memory as possible. At this stage in its development, the Altair microcomputer only had 4K of system memory to work with (a tiny amount). Gates studied the instruction set for the Intel 8080 microprocessor carefully, and he was determined to fit the new BASIC into available memory *and* leave a little room for the user's programs.

How did he go about solving this problem?

We know more about Gates's solution than we do many of his contemporaries because the program that he built became well known, and Gates wrote about his method in two texts that have attracted the attention of journalists. The first comments about his approach were published in September 1975, followed by a longer interview with Susan Lammers about Gates's coding procedures in 1986.[1] I am especially interested in Gates's advice to new programmers of the Intel 8080 microprocessor, because I am fascinated with stories about how novices learn to program. In this case, we can observe a 19-year-old coding prodigy teaching others how to code.

Gates suggested that the best way to familiarize yourself with a new instruction set was "to go out of your way to use every instruction at least once."[2] As you are learning the syntax and architecture of a new chip, Gates advised, "go through the instruction set... and look closely at the instructions you seem to use very rarely." He suggested that successful programmers will continually search for better commands and more efficient ways to solve their problems. But in the final routines, when efficiency is the most pressing concern, it was important to flip the strategy and "use the least number of instructions possible to perform each function."

1. For the sources, see Bill Gates, "Software notes," in *Computer Notes*, ed. David Bunnell (September 1975). Also printed in "Appendix/Bill Gates," *Programmers at Work*, ed. Susan Lammers (Redmond, WA: Microsoft Press, 1986), 354. The 1986 interview with Gates appears on pp. 70–90 of *Programmers at Work*.

2. Gates, "Software notes," 354.

Finally (and I love this recommendation as a technical writer), Gates cautions that one should *not trust the instruction books on programming too much*, because they sometimes neglect important shortcuts.[3] Programmers, in other words, should learn by doing, internalizing every aspect of the instruction set and hardware features until they are deeply engrained in the coding psyche. This sentiment would eventually become a mantra of the learn-to-program movement.[4]

As the team of students completed their work, Bill Gates wrote out the assembly language routines for the interpreter on yellow legal pads, drawing explanatory charts to document what was happening in computer memory.[5] Paul Allen and Monte Davidoff also collaborated at this stage, competing with Gates to make the code as tight as possible. At one point, Gates called owner Ed Roberts at MITS to ask how the Altair processed characters typed on a keyboard, because they had no access to the actual Altair device.[6] Finally, the group typed the program into Harvard's DEC PDP-10 minicomputer using a console display and a keyboard. The PDP-10 was running a software emulator program designed to mimic the Altair microcomputer, created by Paul Allen. When Gates, Allen, and Davidoff finished inputting and debugging their program, they tested its operations by keying in several BASIC programs—the types pioneered by Kemeny and Kurtz at Dartmouth College over a decade earlier.

The programmer-entrepreneurs then used the PDP-10 to create a spool of punched paper tape containing the completed BASIC interpreter. (See Figure 3.1.) Allen flew to Albuquerque, New Mexico, and fed the punched tape into the Altair test machine at MITS, establishing what would become the first commercial high-level programming language for a PC.[7] The entire process took a little over 8 weeks. In the coming years, the nascent Micro-Soft team (later Microsoft) adapted their solution to work on computer systems with different hardware

3. Gates, "Software notes," 354.

4. Gates was also, of course, a self-taught programmer, with little formal instruction in computer science.

5. For samples of the charts and excerpts from the source code, see "Appendix/Bill Gates," in *Programmers at Work*, 348–356. Also useful is the *Altair BASIC Reference Manual* (Albuquerque, NM: MITS, 1975), which contains technical specifications and a command reference for the program that Gates, Allen, and Davidoff created.

6. Michael Swaine and Paul Freiberger, *Fire in the Valley: The Birth and Death of the Personal Computer*, Third Edition (Dallas, TX: The Pragmatic Bookshelf, 2014), 159.

7. This part of the story has been popularized by Walter Isaacson, *The Innovators* (New York: Simon & Schuster, 2014), 332–340; and Swaine and Freiberger, *Fire in the Valley*, 157–161. For an earlier description of Gates's work with the Altair, see Ray Duncan, ed., *The MS-DOS Encyclopedia* (Redmond, WA: Microsoft Press, 1988), 3–19.

Figure 3.2 **Bill Gates and Paul Allen pose for a portrait at Microsoft in 1984. Behind the programmers is a white board with illustrations of computer memory, including a plan for allocating resources in an IBM PC that contains 64KB of RAM.** (Photo by ©Doug Wilson/CORBIS/Corbis via Getty Images)

characteristics, expanding the interpreter's abilities as microcomputers and PCs became more powerful. They applied the same basic approach as they managed system resources for operating systems. (See Figure 3.2.)

As this example demonstrates, there is more than meets the eye to building non-trivial computer programs, and much of the process takes place well *before* the programmer loads the code into memory and actually runs the program. The emphasis here is on *teamwork*, and it serves as a corrective to the misconception that programming is usually the work of a solitary coder sitting alone in front of a computer screen.

Conceptually, programming involves refining *algorithms*, the ordered collections of steps that are proposed to automate processes and solve problems elegantly and efficiently. Some algorithms are limited in scope, like the eight or ten steps that might be necessary to receive contact information from a user and store it in a computer file. (To complete this task, an algorithm might prompt the user for a name and address, assign the input to temporary variables, check the variables for suitable content, format the content, and then insert the information into a database

at the appropriate location.) Algorithms can also be incredibly complex, such as the comprehensive searching and sorting schemes that Google uses to sift through data gathered from the World Wide Web, then present this information to a user via a commercial web browser.

Systematic attempts to teach programming must somehow train students to become proficient in coding skills, the use of algorithms, debugging techniques, and other important abilities. To appreciate how this relatively obscure problem-solving process became a popular movement, we turn now to the proliferation of programming languages in the 1950s, and the development of an extremely successful programming language, FORTRAN.

3.2 The Tower of Babel

In 1981, computer programming pioneer Jean Sammet observed that, by her count, there were already some thousand computer languages in the U.S.[8] The proliferation of languages was not new, nor was it tied to the development of PCs. In fact, 20 years earlier there were already so many languages in use on mainframe computers that the journal *Communications of the ACM* published a "Tower of Babel" image on its January 1961 cover. (See Figure 3.3.) The image depicted the mythical tower-to-heaven structure described in the biblical book of Genesis, glossed with the names of dozens of computer languages on the mythical tower's rings. The artwork recalls earlier critiques of "progress" in America's social and political history, and it may also poke fun at the hubris of software industry officials for propagating so many compilers. (Human hubris is a pressing concern of the Genesis narrative.)

In fact, the task of learning to program has sometimes been explained as a simple process of picking a computer language and learning all its features, as if mastering a language's grammar is the same thing as learning to think logically, or to understand how a computer processes information. As we observed earlier, however, software development involves much more than simply learning the syntax of instructions, as important as that may be. Jean Sammet captured the importance of language syntax when she commented: "In the last analysis [language choice] almost always boils down to a question of personal style or taste."[9] In other words, the syntax of languages is interesting, but much in the differences between systems is simply a matter of fashion or technical culture.

8. Sammet cited in Nathan Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise* (Cambridge, MA: The MIT Press, 2010), 102. For the context of Jean Sammet's work with languages, see Richard L. Wexelblat, ed., *History of Programming Languages* (New York: Academic Press, 1981).

9. Jean E. Sammet, "Programming languages history," *Annals of the History of Computing* 13, no. 1 (1991): 49.

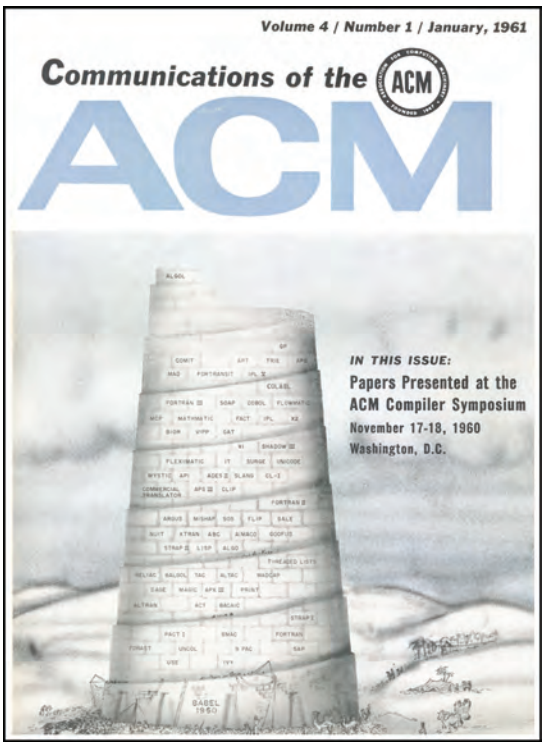


Figure 3.3 ***Communications of the ACM*** **"Tower of Babel" Cover Image (January 1961), depicting the multiplication of computer programming languages** (Courtesy of the ACM)

So where did all the languages come from and why do programmers propagate them?

As I noted earlier, the first electronic computers did not utilize software or what we now call "programming" at all. They were hardwired devices that performed individual tasks, such as calculating the trajectory of a rocket. If you wanted to change the problem being computed, you didn't modify the software, you changed the wiring to accommodate the problem. By hand.

An example of this type of device is the so-called Atanasoff–Berry Computer, conceived in 1937 to solve linear equations, one problem at a time. The computer was assembled over a 5-year period at Iowa State College. When it was finished, the machine could be set up to solve two linear equations with up to 29 variables. This was impressive work and the results were highly valued by the Iowa State Physics Department. But in this context the device was essentially a single-purpose computer that specialized in linear equations.

One of the first *programmable* computers was the ENIAC, designed by John Mauchly and J. Presper Eckert at the University of Pennsylvania. Dedicated in 1946, the ENIAC utilized sophisticated wiring, 18,000 vacuum tubes, panels of switches, and punched-card equipment for input and output.

The physical task of programming the ENIAC was considered less important than the abstract task of devising complex numerical calculations that the machine could solve. Accordingly, the job of "setting up" the computer with punched cards, cables, and switches was left to skilled female workers who had been trained in mathematics. (Note: The ENIAC team used different terminology than later computer designers, so the terms "programming" and "programs" in this section are somewhat anachronistic.)[10]

Regardless of gender considerations, it was not easy to create actual programs for the ENIAC system. In addition to conceptual errors and coding mistakes that arose as part of the planning process, the early log books indicate that there were regular shut downs due to faulty tubes, short circuits, carry errors, divider faults, water leaks, and other problems.[11] In short, programming involved a host of physical issues in the early days that modern software developers have no knowledge of or responsibility for today.

In the 1950s, most early programs were written in numerical machine code, which consisted of 1s and 0s representing instructions for a specific computer. Programmers needed to learn the instruction set for a given computer, and then express the instructions in binary (base-2), octal (base-8), or hexadecimal (base-16) number systems depending on the machine's internal architecture. Octal was especially common in early computer systems like the DEC PDP-8, ICL 1900, and the IBM mainframes, which structured internal memory using 12-bit, 24-bit, and 36-bit words.

Grace Murray Hopper described the intricacies of designing programs in this era in a keynote address at the first ACM conference on the history of programming languages, held in 1980 to document the achievements of the early days. (Hopper is shown with a Univac I computer system in Figure 3.4.) At the conference, Hopper explained that she wrote machine code programs in octal, where she routinely added, subtracted, multiplied, and divided in base-8 arithmetic. Hopper performed routine mathematical calculations in her head as she completed her

10. For an excellent analysis of the ENIAC computer and those who designed and operated it, see Thomas Haigh, Mark Priestley, and Crispin Rope, *ENIAC in Action: Making and Remaking the Modern Computer* (Cambridge, MA: The MIT Press, 2016). Most of the calculations were numerical in nature, such as plotting the trajectory of a rocket or ballistic calculations.

11. Haigh et al., *ENIAC in Action*, 79–83.

Figure 3.4 **The operator's console of a Univac I computer with four computer programmers (1957). From left to right, Donald Cropper, K. C. Krishnan, Grace Murray Hopper, and Norman Rothberg.** (Courtesy of the Computer History Museum)

work, although it was also common for engineers to use lookup tables to save time. (Ironically, Hopper later found it difficult to balance her own checkbook using base-10 arithmetic, as she was so steeped in using octal.)[12] Later, Hopper and her peers used assembly language when newer computers arrived as way to write computer instructions in a more readable (textual) format.

In assembly language, program instructions are composed using short names or abbreviations (mnemonics) for machine language instruction codes. For example, the instruction "ADD" instructs the central processing unit to add the contents of one register to another. Symbolic names are also used to reference memory locations in assembly language program code.

Most computer science students learn assembly language as part of their introduction to machine architecture, even if they don't go on to program exclusively

12. Grace Murray Hopper, "Keynote address," in *History of Programming Languages*, ed. Wexelblat, 7.

in this language. In my college years (the early 1980s), I started with the MACRO assembly language for the popular DEC VAX line of minicomputers.[13] Later at Microsoft, I used Microsoft Macro Assembler (MASM) for programming IBM PCs and compatibles running MS-DOS. Coding skills of this type were standard fare in 1980s' programming culture as both hobbyists and professionals needed to squeeze as much power and efficiency from the early systems as possible. (For more about MS-DOS programming requirements and the architecture of IBM PCs and compatibles, see Chapter 6 and Chapter 9.)

Assembly language is a nice improvement over machine language, but it is still closely connected to the hardware architecture of a computer system. Assembler is a *low-level* programming language designed for speed. This means that assembly language programs written for one computer will only work on computers of the same model or type. Translating an assembly language program from one computer model to another invites a lot of work, and this made it difficult to devise general-purpose programming solutions that could be deployed on multiple systems. However, as we learned with the Altair programming example, assembly language is valuable when you want to master a chip's instruction set and create compact and efficient code. This is probably why computer scientist Donald Knuth (1938–) wrote so many of his algorithms in machine (or assembly) language in his magisterial book series, *The Art of Computer Programming*. As a rallying cry for efficient code, Knuth advised:

> High-level languages are inadequate for discussing important low-level details such as coroutine linkage, random number generation, multi-precision arithmetic, and many problems involving the efficient usage of memory. A person who is more than casually interested in computers should be well schooled in machine language, since it is a fundamental part of a computer.[14]

Donald Knuth wrote this in the foreword to his popular book series in 1962, and it was reprinted in many editions. His writings and advice are highly valued by academics and self-taught programmers alike.

13. My assembly language textbook was James Brink and Richard Spillman, *Computer Architecture and VAX Assembly Language Programming* (Menlo Park, CA: The Benjamin/Cummings Publishing Company, Inc., 1987). The authors were my instructors at Pacific Lutheran University. For a sample computer book that teaches MASM, Microsoft's assembly language system for IBM PCs and compatibles, see Ray Duncan, *Advanced MS-DOS Programming*, Second Edition (Redmond, WA: Microsoft Press, 1988).

14. Donald E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Third Edition (Berkeley: Addison-Wesley, 1997), ix.

But were there even earlier attempts to teach programming to new computer users?

Arguably the world's first computer book containing specific instructions about how to program a computer was published in the U.K. in 1951. This impressive volume was entitled *Preparation of Programs for an Electronic Digital Computer*, and it was concerned with formulating machine code for the revolutionary electronic delay storage automatic calculator (EDSAC) computer at the University of Cambridge.[15] (See Figure 3.5.) The authors were Maurice Wilkes, David Wheeler, and Stanley Gill, pioneering professors and technical writers connected with the Cambridge community. Their book offered a selection of common subroutines for handling basic operations in a computational program. The instructions were specific to the EDSAC, one of the world's first stored-program computers.

The authors undertook daunting challenges, because no programming book had been written before and there was little in the way of notation or written conventions to express complex step-by-step instructions in book form. In fact, the routines are not "code" at all but instructions on how to set up the machine's registers and the parameters needed to solve certain types of equations. But by distributing a selection of common "routines," the authors explained to fellow scientists how the EDSAC worked, and readers with access to a similar computer could use the book to save time and expand their programming skillset. From this began a venerable tradition that continues in all good programming primers up to the present—readers learn by example.

Preparation of Programs did well enough such that a second edition was prepared in 1957. In the revised edition, the authors provided routines that were adaptable to a wider range of stored-program computers. In a literary sense, the "learn-to-program" movement began with these books, although the intended audience was a narrow band of scientists and engineers and not the general public.

3.3 High-level Languages

Beginning in the 1950s, Grace Hopper and her peers began to develop a solution that would make software less costly to produce and programming easier to learn.[16] This solution was known to contemporaries as an "automatic programming language" or "autocode" for short. We know it today as the first of many *high-level languages*—software abstractions with a syntax closer to human language—which

15. Maurice Wilkes, David Wheeler, and Stanley Gill, *Preparation of Programs for an Electronic Digital Computer* (Reading, MA: Addison-Wesley, 1951).

16. A useful starting place to survey Grace Murray Hopper's fascinating career is Kurt W. Beyer's *Grace Hopper and the Invention of the Information Age* (Cambridge, MA: The MIT Press, 2012).

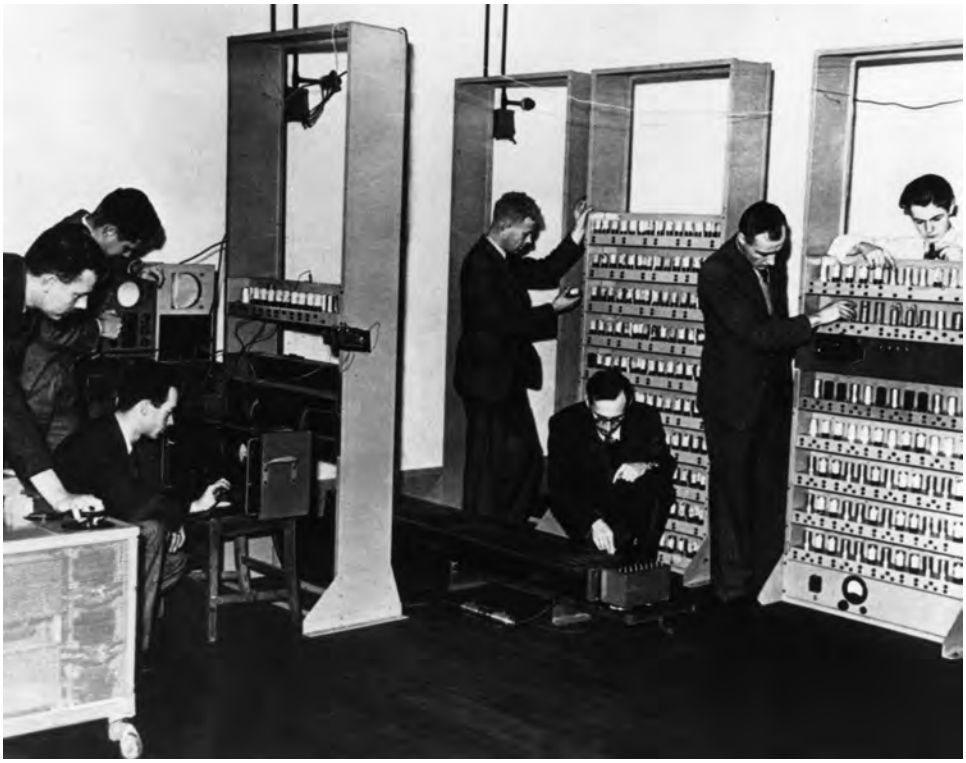


Figure 3.5 **Maurice Wilkes and his colleagues work on the EDSAC computer at the University of Cambridge, U.K. Wilkes is in the middle, kneeling and wearing glasses.** (Courtesy of the Computer History Museum)

hide some of the inner-workings of a computer. Using a high-level language, a programmer can write instructions using recognizable statements and symbols, then use a program known as a *compiler* to translate the high-level statements into the machine code required by the underlying computer. Hopper wrote her first compiler between October of 1951 and May of 1952, and it was called the "A-0 Compiler."[17] This proved that higher level languages were possible and useful, and scientists began designing them in earnest. In the coming years, Hopper's work became the foundation for the FLOW-MATIC and COBOL languages.

At first, the high-level compilers created machine code that was less efficient than human-generated machine code, but as time passed the compilers improved, and the benefits of high-level abstraction became obvious. For one thing, higher-level languages saved programming time, as software developers could

17. Hopper, "Keynote address," 10.

more quickly build a solution if they didn't have to manage operations inside the computer at a minute level.[18] Software became easier to revise and update in future releases, and the code was easier to share among team members. In addition, high-level language designers could customize their coding systems so that they met the needs of a specific industry or computer application. For example, the language's instructions, keywords, and data structures could be readily adapted to such tasks as numerical analysis, list processing, artificial intelligence, music, and education.

Soon, there were dozens—even hundreds—of high-level computer languages. A standard task in graduate Computer Science programs became developing new compilers as an exercise to learn advanced language concepts, such as pattern recognition, lexical analysis, code optimization, and object-oriented programming. New computer science textbooks arrived in the 1970s and 1980s to introduce this fundamental skill and its applications.[19]

The most important early high-level languages were FORTRAN, COBOL, ALGOL, BASIC, Pascal, and C. A great amount of documentation and books related to these languages and their early supporters is available in print and electronic media, waiting for the historians of computing to assess them. These sources include "founding memoirs" written by language pioneers, early language specifications and manuals, primers, corporate marketing materials, journal articles written for computing professionals, product reviews, and so on. Overlooked in the study of these languages is the abundant supply of primers or "how-to" computer books written for students, self-taught programmers, and hobbyists who sought to learn programming on their own terms or study together in a classroom or user group setting. Throughout *Code Nation*, I argue that computer books and magazines are a vital but neglected source of information that will help historians and computer scientists understand how technical ideas and techniques were diffused from inventors and engineers to the general public. These historical sources are just as important as "founding memoirs" for understanding how programming culture took shape in the U.S. Programming primers are in essence the daily newspapers and chapter

18. The magnitude of this breakthrough and its importance in the history of computing is skillfully summarized in Paul Ceruzzi, "An unforeseen revolution: computers and expectations, 1935–1985," in *Imagining Tomorrow: History, Technology, and the American Future*, ed. Joseph J. Corn (Cambridge, MA: The MIT Press, 1986), 188–201, here at 199.

19. Computer science students in the 1980s and 1990s learned about the world of compilers and their histories from books like Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools* (Reading, MA: Addison-Wesley, 1988). We called this the "dragon book" because it featured a bright orange dragon on the front cover, emerging from a computer screen.

books of the "PC Revolution"; they contributed significantly to the learn-to-program movement's propagation across American society.

To open a window on the movement's origins in the 1950s and 1960s, let's examine the foundations of FORTRAN, arguably the most important platform for programming activity in the 1960s. FORTRAN products and publications encouraged new users to experiment with software creation. These computer users were often proficient in science and mathematics but not yet familiar with how computers worked. Once FORTRAN built momentum and established an audience, the learn-to-program movement expanded into other popular languages like COBOL, BASIC, Logo, and Pascal.

3.4 Learning FORTRAN

FORTRAN was initially designed by John Backus and a group of research scientists at IBM. (See Figure 3.6.) The team designed its high-level language in 1954, and then they took some 2.5 years to build the FORTRAN "translator," as the organization described their compiling system that converted FORTRAN routines into machine code.

IBM released FORTRAN I in April of 1957, and the language and compiler quickly became a standard for software development in the U.S. and Europe. Important for its adoption, IBM distributed the original FORTRAN software and materials for free to its customers who owned IBM hardware, eventually attracting the interest of a standards organization that built consensus around the new language and its features.

The term "FORTRAN" looks like an acronym, but the name is actually an abbreviation for the words "formula translation" or sometimes "formula translating system." True to its original names, the language was created for solving scientific and engineering problems, and its internal structures were designed to process mathematical equations in a straightforward way. The language included standard mathematical functions, conditional statements, looping structures, variable types, arrays, and other features that would become customary in high-level languages. All variables beginning with the letter I, J, K, L, M, or N were automatically declared as "fixed point" (or integer) values. All other variables were typed as "floating point" (or real) numbers. The language also allowed the programmer to insert comments into his or her code, allowing developers to explain what their routines were doing and where their code still needed work.

When IBM designed FORTRAN, it was deeply concerned with improving programmer productivity. Large software development projects were already developing a reputation for being over time and over budget, and this "crisis" gained major notoriety in the 1960s. Reflecting on the economics of this issue as early

Figure 3.6 **Gathering of the engineering team that contributed to IBM FORTRAN in the 1950s. (From the 'Pioneer Banquet' at the National Computer Conference, Houston, Texas, 1982.) From left to right: Richard Goldberg, Robert Nelson, Lois Haibt, Roy Nutt, Irv Ziller, Sheldon Best, Harlan Herrick, John Backus, and Peter Sheridan.** (Courtesy of the Computer History Museum)

as 1954, Backus wrote: "Programming and debugging accounted for as much as three quarters of the cost of operating a computer; and obviously, as computers got cheaper, this situation would get worse."[20] Early results with the new compiler were promising, however. When compared to low-level languages, FORTRAN programmers often reduced the number of statements that they used to write their code by a factor of 20.

But what did it mean to "write code" in the early days of FORTRAN?

In the first batch-processing systems, FORTRAN programmers did not use text editors to write their programs, but they wrote out routines by hand and then entered them on a key punch device that produced 80-column punched cards. The contents of one FORTRAN program statement appeared on each card, though longer statements could span multiple cards. To help with the process of preparing the cards, IBM produced standard coding sheets for programmers to fill out that identified the columns, statement numbers, and FORTRAN language elements. If the programmer checked a special box at the top of the card, the card's contents

20. Jim Backus, "The History of FORTRAN I, II, and III," in *History of Programming Languages*, ed. Wexelblat, 26–27.

would be considered a "comment", i.e., an explanatory description ignored by the compiler but retained to document what the program did.

Any spaces existing in the final program would also be ignored by the FORTRAN compiler. Spaces were known to cause confusion because they were hard to discern on the coding sheets—both for programmers and key punch (or card punch) operators. Jim Backus later wrote that his team was criticized for designing the compiler so that it ignored spaces, but an allowance of this type was typical in an era where programmers worked in large teams that had different levels of training. Many needed to use hand-written notes (as well as cumbersome mechanical devices) to accomplish their work, and in these contexts, errors crept in.[21]

A complete FORTRAN program consisted of a deck of punched cards that a technician could feed into a card reader, which was attached to the mainframe computer. Once the program was loaded into memory it could be compiled, and the technician would receive a report if there were any errors. The process of debugging the program then began, which could take some time to complete and might involve many team members. In batch-processing contexts like these (i.e., before interactive terminals with keyboards), it might take days or weeks to fix a relatively simple logic or runtime error in a program.

My favorite textbook describing this process for new programmers is Marshal H. Wrubel's *A Primer of Programming for Digital Computers* (1959).[22] This well-written book includes step-by-step instructions for filling out coding sheets, creating punched cards, loading cards into an IBM 650 system, and then testing programs using a variety of methods. There is also a fascinating chapter comparing the early FORTRAN, IT, and FORTRANSIT compilers—all designed to help scientists solve math problems. Marshal Wrubel (1924–1968) was a Juilliard piano prodigy and a University of Chicago astrophysicist who took up computing in the 1950s and eventually ran the Research Computing Center of Indiana University. He is relatively unknown in the history of computing because he died unexpectedly at the age of 42 on a hike in the mountains of Colorado. Sadly, he published this innovative primer but nothing more about programming. However, he made an important contribution to technical writing with this title.

The first official manual to document the FORTRAN compiler arrived in October 1956, when IBM released the *Programmer's Reference Manual*.[23] This formal guide

21. Backus, "The History of FORTRAN I, II, and III," in *History of Programming Languages*, ed. Wexelblat, 32.

22. Marshall H. Wrubel, *A Primer of Programming for Digital Computers* (New York: McGraw-Hill, 1959).

23. IBM, *Programmer's Reference Manual: The FORTRAN Automatic Coding System for the IBM 704 EDPM* (New York: IBM Corporation, 1956).

offered a short introduction to FORTRAN and gave experienced programmers the information they needed to use the compiler on the IBM 704 system. The book carefully documented each new language feature, highlighting the language's keywords and grammar, or what programmers called *statement syntax*. The manual also included a short chapter on how to create solutions for two "sample problems."

Soon after this, a selection of user-friendly primers teaching FORTRAN began to appear. These introduced the language to a broader audience (i.e., not just IBM customers). These books included Grace E. Mitchell's *Programmer's Primer* (1957), Daniel McCracken's *A Guide to FORTRAN Programming* (1961), and Elliott Organick's *A Primer for Programming with the FORTRAN Language* (1961).[24] These books demonstrate that the nascent programming movement was spreading beyond research labs and academic contexts into settings where quantitative calculations were part of everyday work.

As primary sources for historians, the first FORTRAN primers provide evidence of how coding was gradually seeping into public consciousness. Students, business people, and engineers were now the audience for these books, rather than IBM customers who were using just one system. The computer books also document literary developments in technical writing, i.e., the use of a less-formal "author voice" in scientific descriptions. Some of the books felt open-ended, engaging a wider audience. The tutorials also benefited from the gradual shift from batch-processing to time-sharing, which allowed for greater access to computers and a more interactive experience.

Grace E. Mitchell's *Programmer's Primer* was among the most influential books of this era for budding technical writers. (See Figure 3.7.) Although Mitchell was uncredited in the text and the book was published as an IBM reference manual, it was just as pioneering as the commercial tutorials published by mainstream publishers.

Mitchell joined IBM's FORTRAN group in the Spring of 1957, and she made important contributions to the FORTRAN II compiler release, as well as IBM's newest operating systems and programming tools. Although Mitchell possessed formidable engineering skills, her *Primer* was designed for non-programmers who were preparing to take their first steps with a new language. In reality, readers *did* need some knowledge of college algebra, trigonometry, and matrix operations to

24. IBM [Grace E. Mitchell], *Programmer's Primer for FORTRAN: Automatic Coding System for the IBM 704 Data Processing System* (New York: IBM Corporation, 1957, revised ed., 1958); Daniel McCracken, *A Guide to FORTRAN Programming* (New York: Wiley & Sons, 1961); Elliott Organick, *A Primer for Programming with the FORTRAN Language* (Houston, TX: Computing and Data Processing Center, 1961; [Addison-Wesley edition, 1963]).

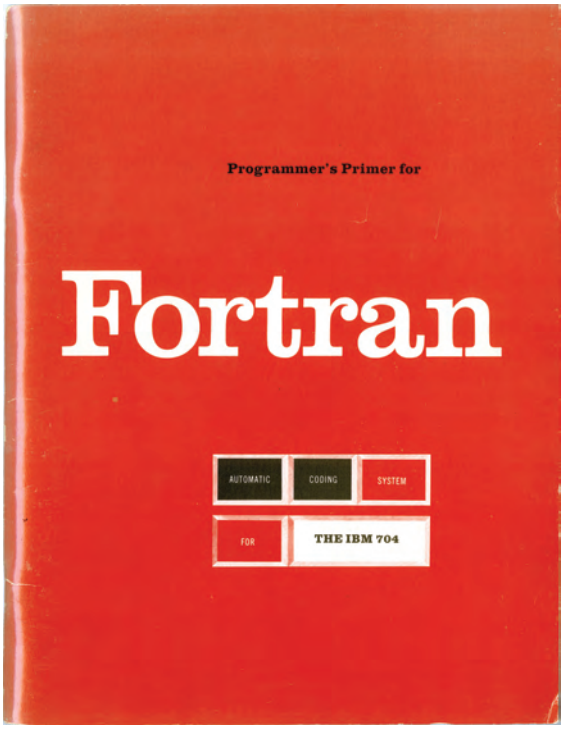
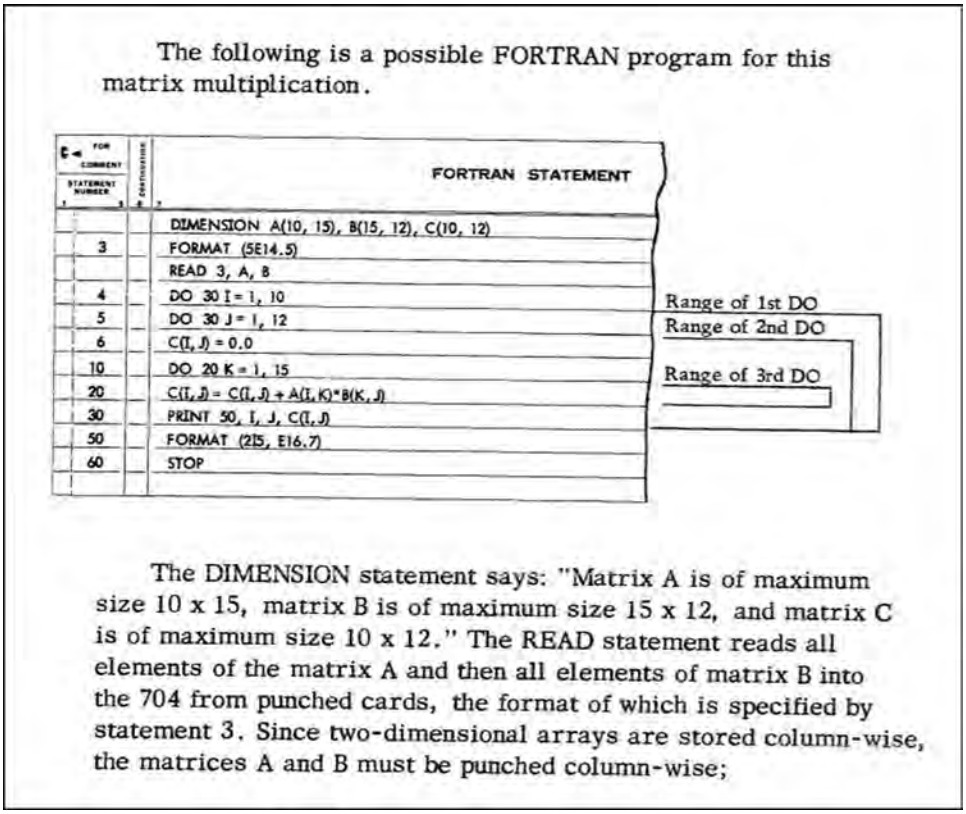


Figure 3.7 **Programmer's Primer for FORTRAN: Automatic Coding System for the IBM 704 (1957). Grace E. Mitchell was the uncredited author of this early "how to" programming tutorial from IBM.** (Courtesy of the Computer History Museum)

make much use of this book. But no specific computer knowledge was required, and Mitchell assumed that programming would be an entirely new concept. In 1957, most of IBM's FORTRAN users were scientists who were well-trained in engineering but knew little about computers.

Mitchell's primer offered a concise analysis of FORTRAN's core features, including some of the complexities only hinted at in the original *Reference Manual*. (See Figure 3.8.) For example, Mitchell included a lengthy section on working with two- and three-dimensional arrays in FORTRAN I. The tutorial explored how to declare arrays, assign initial values, and step through arrays using DO loops and other control structures.[25] Mitchell also included practical information about debugging unruly programs, a topic that was largely neglected in the *Reference*.

25. IBM, *Programmer's Primer for FORTRAN*, 43–64.



The following is a possible FORTRAN program for this matrix multiplication.

STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
		DIMENSION A(10, 15), B(15, 12), C(10, 12)
3		FORMAT (5E14.5)
		READ 3, A, B
4		DO 30 I = 1, 10
5		DO 30 J = 1, 12
6		C(I, J) = 0.0
10		DO 20 K = 1, 15
20		C(I, J) = C(I, J) + A(I, K)*B(K, J)
30		PRINT 50, I, J, C(I, J)
50		FORMAT (2I5, E16.7)
60		STOP

The DIMENSION statement says: "Matrix A is of maximum size 10 x 15, matrix B is of maximum size 15 x 12, and matrix C is of maximum size 10 x 12." The READ statement reads all elements of the matrix A and then all elements of matrix B into the 704 from punched cards, the format of which is specified by statement 3. Since two-dimensional arrays are stored column-wise, the matrices A and B must be punched column-wise;

Figure 3.8 **Excerpt from Mitchell's *Programmer's Primer* (1957, page 44), showing how to write a FORTRAN program that computes matrix multiplication. The FORTRAN code was displayed on a standard coding sheet, making it easier to identify the columns, statement numbers, and language elements needed.** (Courtesy of the Computer History Museum)

3.5 Daniel McCracken's Primers

Daniel D. McCracken (1930–2011) also established his reputation as a programming author, publishing some of the first general textbooks about software development in the U.S. (See Figure 3.9.) McCracken wrote his first computer book at the age of 24 while working for General Electric, a 256-page text entitled *Digital Computer Programming* that appeared during the fall of 1957.[26] This title was published by John Wiley & Sons, one of the first computer book publishers. Wiley went on to develop an international reputation for reliable computing titles in several categories. The influential acquisitions editor at Wiley who worked with

26. Daniel D. McCracken, *Digital Computer Programming* (New York: John Wiley & Sons, 1957).

Figure 3.9 **Former ACM President and computer book author Daniel D. McCracken.** (Image courtesy of the Charles Babbage Institute, University of Minnesota Libraries)

McCracken was Walker Stone, a determined technologist who had taken a few programming courses in the 1950s and quickly recognized the computer's importance and value.[27] Stone oversaw technical publishing at Wiley for several years, establishing the Information Science Series and other well-regarded imprints.

Looking back at his career, McCracken said that he wrote the Wiley book in 1957 because there was no textbook available for programmers, and he wanted to write one that described the features of a "hypothetical machine," which he conceived of as a cross between an IBM 704 and an IBM 650.[28] Using this platform, McCracken introduced students to basic programming concepts like looping, branching, input, output, and floating-point arithmetic. The textbook had a relatively limited appeal because of its narrow audience (at the time, his book competed with the second

27. Daniel McCracken, interview by Arthur L. Norberg, January 7–9, 2008, Charles Babbage Institute, University of Minnesota, 17–18.

28. McCracken, interview by Arthur L. Norberg, 15.

edition of Maurice Wilke's book on EDSAC programming), but McCracken's text did find its way into some of the earliest programming courses in America. For example, Jean Sammet is reported to have used McCracken's book at Adelphi College on Long Island soon after its release.[29]

McCracken went on to publish *A Guide to FORTRAN Programming* with Wiley in 1961, a textbook that eventually sold 300,000 copies. This title has been described as the first-bestselling book for programmers in the U.S., an achievement related to McCracken's acumen as a writer, Wiley's skillful marketing, and the emergence of FORTRAN as *the* standard high-level language. McCracken described the magical combination several years later: "Walker Stone saw it, convinced management, and I just turned all possible effort to getting a FORTRAN book out fast before anybody else got the idea. I wrote that book in under six months."[30]

McCracken's FORTRAN tutorial came in at just 88 pages, and it was published in a flexible workbook format (8.5" × 11") to encourage student use and allow for propping the book open in front of a computer terminal. The textbook's success contributed to the rising tide of FORTRAN books, broadening its appeal. McCracken also made programming seem interesting and approachable, distinguishing his author voice from that of the FORTRAN manual. Ken Thompson (see Figure 10.1), the creator of Unix, later described the importance of learning FORTRAN in this way: "95 percent of the people who programmed in the early years would never have done it without FORTRAN. It was a massive step."[31] Over time, FORTRAN dramatically increased the productivity of America's engineering teams (see Figure 3.10).

But McCracken wasn't finished. By the end of the 1970s, McCracken had become the author or co-author of over two dozen computer books which sold more than 1.6 million copies and were translated into 15 languages. He served as Vice President of the ACM from 1976 to 1978, and President of the ACM from 1978 to 1980. Although his work had an important impact on corporate and scientific computing, his commitment to social issues was equally significant. For example, he led an effort to develop social statements on the Vietnam War, the Equal Rights Amendment, human rights, and privacy issues.[32] Considering his legacy as a bestselling author of computer titles, the *New York Times* simply described McCracken as the

29. McCracken, interview by Arthur L. Norberg, 15.

30. McCracken, interview by Arthur L. Norberg, 18.

31. Kenneth Thompson, "When few knew the code, they changed the language," *New York Times*, June 13, 2001.

32. McCracken was also the first chairman of the ACM's Committee on Computers and Public Policy, established in 1973. See Janet Toland, "'Deeply Political and Social Issues': Debates within ACM 1965–1985," in *Communities of Computing: Computer Science and Society in the ACM*, ed. Thomas J. Misa (San Francisco, CA: Morgan & Claypool/ACM Books, 2017), 111–141.

Figure 3.10 **An engineering team at Digital Equipment Corporation pose in this undated photo. FORTRAN's success dramatically increased the productivity of development groups.** (Courtesy of the Computer History Museum and DEC)

"Steven King of how-to programming books."[33] What is less well known is that McCracken used a portion of his royalties to fly around the country advocating for political and social issues.[34] It was a furtive period of change in which peace activists mixed with research scientists, authors, and programmers.

How broad was the appeal of the nascent learn-to-program movement?

33. Steve Lohr, "Daniel D. McCracken, expert on computers, dies at 81," *New York Times*, August 15, 2011, B8.

34. Toland, "'Deeply Political and Social Issues,"' 115.

The ACM was an active participant from the start. In 1961, ACM luminary Alan Perlis offered what may have been the first *universal* appeal for college students to learn how to program. Perlis suggested that *all* university freshmen take a programming class at the Carnegie Technical Institute to learn about computers and how they operated. His article, "The role of the digital computer in the university," was published in the popular technical journal, *Computers and Automation*.[35] Although Perlis preferred ALGOL, a concise high-level language that he co-developed, he was less interested in syntax and more interested in the patterns of computational thinking. Professor Perlis was highly influential in how Computer Science was introduced in the universities, and his advice counted. He went on to join the ACM Computing Curriculum Committee, an influential standards body that proposed the first curriculum standards for Computer Science education in the mid-1960. The ACM also formed the Special Interest Group on Computers and Society (1969) in part to advocate for computer literacy.

3.6 Seymour Papert and Logo

Through FORTRAN, programming instruction became one avenue into the world of computing in the 1960s, encouraged by the rising tide of primers and other materials. But not everyone agreed on which learning system should be used or what computational literacy might entail in schools. Moreover, the growing accessibility of computers brought up intellectual and philosophical questions. Might computational thinking be valuable for its own sake, outside of professional contexts? Might learning to code support cognitive development in children and adults? What, ultimately, was the purpose of programming instruction?

At MIT, an innovative group of artificial intelligence (AI) researchers began to propose intriguing answers. Their efforts would produce not only a new programming language, but educational strategies that would profoundly influence how Americans taught programming for years to come. The leader of this group was Seymour Papert (1928–2016), a mathematician and psychologist who co-founded MIT's AI laboratory with Marvin Minsky (1927–2016). Papert co-developed the Logo programming language and launched what became known as the constructionist movement in science education.

After receiving a Ph.D. in Mathematics at the University of Cambridge (1959), Papert studied for 5 years with psychologist Jean Piaget (1896–1980) at the Center for Genetic Epistemology in Geneva, Switzerland. Papert came away impressed by Piaget's way of seeing children as active builders of their own intellectual structures.

35. Alan Perlis, "The role of the digital computer in the university," in *Computers and Automation* 10, 4 and 4B (1961): 10–15.

Papert came to believe that children readily used the materials *that they found about them* to learn, and that these tools were most efficacious when they were a regular part of the surrounding culture.[36] For Papert, such a tool could be the computer, if it could be adapted to the educational aims of teachers and the natural experience of students.

Papert and his colleagues began to ask important new questions about learning to use computers in public settings. How might computers affect the way that people think and learn? Can computers be carriers of powerful ideas and the seeds of cultural change? How can computers help people form new relationships with knowledge that cut across the traditional lines separating science from the humanities? Can people of all ages learn computing principles? If the goal of teaching about computers is cognitive development, what is the ideal age to start young children?

Papert was particularly interested in using computer *programming* to enhance cognitive development. However, the psychologist was unimpressed with the way that people were learning programming skills in his day, and he believed that the current crop of programming tools and primers were only suited for adults. A mathematician by training, Papert believed that computers could revolutionize math instruction if coding tasks were more naturally connected to a child's developmental impulses. His MIT group responded by creating Logo, a high-level language and system that utilized visual output (computer graphics) and automated devices (robotics) to teach programming. The Logo language was co-developed by Seymour Papert, Cynthia Solomon, and Wally Feurzeig in 1966, and the system made its debut in Cambridge, Massachusetts the following year. Within a decade, Logo became the leading educational computer language in the U.S., rivaled only by BASIC in high schools and Pascal in university settings.

Logo's most iconic learning feature was the *turtle*, an on-screen shape that showed the result of the language's movement commands. Some implementations of Logo also featured an animated Turtle Writer robot (see Figure 3.11), which students could program to draw shapes and move around the classroom. (The robots were also equipped with sensors that allowed them to avoid obstacles and learn from their environment.) When programmers typed in Logo commands at a terminal console (or later, on a PC keyboard), the commands produced line drawings and other shapes that the turtle could create—the so-called "turtle graphics."

Because Logo is an interpreted language, each command is executed as soon as the programmer enters it, and users are able to see the output of their program statements immediately. Papert and his colleagues used this interactivity

36. Seymour Papert, *Mindstorms: Children, Computers, and Powerful Ideas* (New York: Basic Books, Inc., 1980), 19.

Figure 3.11 **Seymour Papert at MIT with a Turtle Writer robot and a fish shape that it produced.** (Image courtesy of the Computer History Museum)

to create a hands-on world of block-building and experimentation that (for them) characterized early childhood development, particularly formative experiences with spatial reasoning and mathematics. Later on, more conceptual programming concepts were introduced with additional language features. In fact, the MIT team highly valued the "pre-programming" tasks associated with computer programming—the problem solving, algorithm building, and skill self-assessment that they believed promoted deep learning. Papert found the process of testing and debugging programs to be particularly instructive, because it gave students immediate problem-solving practice when things went wrong.

Papert's goal was to make computer programming an immersive process. In his book *Mindstorms*, he described his objectives in relation to human language learning:

> It is possible to design computers so that learning to communicate with them can be a natural process, more like learning French by living in France than

> like trying to learn it through the unnatural process of American foreign-language instruction in classrooms…
>
> The idea of "talking mathematics" to a computer can be generalized to a view of learning mathematics in "Mathland"; that is to say, in a context which is to learning mathematics what living in France is to learning French.[37]

Rather than using a *computer to program the child*, so that the child might learn to mimic the computer's ways, *the child should program the computer*, acquiring a feeling of mastery over the device, developing a sense of agency from intimate contact with the technology.

Papert's ideas had political and economic consequences, because he recognized that American schools had limited access to computers and time-sharing systems in the 1960s. (In fact, it would be challenging to provide children with even limited access to computers through the 1970s and 1980s.) But Papert's concerns about *access* and the social conditions for learning echo calls for universal tools and the "convivial technology" that we observed in the writings of Ivan Illich, Stewart Brand, Lee Felsenstein, and Ted Nelson in Chapter 2. Indeed, although Papert's work is not usually framed as "countercultural," his circles shared many sympathies with countercultural technologists in Europe and the U.S. In the following years, educational specialists in Britain would introduce computers to children in what they called infants school and primary schools. (See Figure 3.12.) There was also an early relationship between the Logo team at MIT and the research group at Xerox PARC in the San Francisco Bay Area. Daniel G. Bobrow wrote the first version of the Logo program in Lisp while working in the AI group at MIT. In 1972, he moved to Xerox PARC and worked there for several decades. Cynthia Solomon also worked for Apple and Atari in the 1980s, overseeing implementations of the Logo language for PCs.

Logo was created in the research labs of MIT and Bolt, Beranek and Newman (BBN), but the language had its greatest impact after PCs made educational software more accessible to students. Planning for a long future with technology was always part of Papert's vision, and he worked to establish pathways between computers and education all his life. "My discussion of a computer culture and its impact on thinking presupposes a massive penetration of powerful computers into people's lives. That this will happen there can be no doubt."[38]

Papert's colleagues, Wally Feurzeig, Cynthia Solomon, and Daniel Watt, were also instrumental in disseminating the group's ideas into the community. Wally

37. Papert, *Mindstorms*, 6.

38. Papert, *Mindstorms*, 23–24.

Figure 3.12 **Five-year-olds Mark Goulden and Nicola Millar work with Cecil, the first microcomputer used in a British infants school (December 1980). During this stage of the learn-to-program movement, policy makers debated about the best age to expose children to programming concepts and computers.** (Photo by SSPL/Getty Images)

Feurzeig (1927–2013) had a 50-year career at BBN in Cambridge, where he specialized in AI research and the interactive use of computers in schools. In the early 1960s, Feurzeig was interested in time-sharing systems and interpreted computer languages, and he envisioned these technologies working together to make learning easier for students. Feurzeig created the TELCOMP computer language in 1964 to teach elementary mathematics through programming, followed by the Stringcomp language that supported algebraic expressions and higher-level concepts. After these efforts, he tested his ideas with the support of the U.S. Office of Education, arranging for programming classes in eight elementary and middle

schools in the Boston area during the 1965–1966 school year.[39] Feurzeig brought language design skills and much practical experience to the Logo group, along with a deep commitment to teaching coding and interactive learning.

3.7 Cynthia Solomon

Cynthia Solomon also worked at BBN and MIT in the 1960s, joining the AI group in 1965. She started her career with a Bachelor's degree in History from Radcliffe College, and later earned Master's and Doctoral degrees in Computer Science and Education, respectively. In *Mindstorms*, Seymour Papert writes that his collaboration with Solomon was so close, and over such a long period of time, that he found it impossible to enumerate all the contributions that she made to enriching education in computational environments.[40] For example, Solomon was the first to develop an intellectually coherent methodology for training teachers to introduce children to computers.

Solomon's research and writing on programming and cognitive development informed a Doctoral thesis at Harvard and the subsequent book *Computer Environments for Children: A Reflection on Theories of Learning and Education* (1986).[41] This scholarly study explores several models for learning to program and using computers, including rote learning techniques (drill and practice), self-expression (the Plato system), trial and error experiments (using BASIC), and the Piagetian learning system (Logo). In appreciation for this work and her pioneering efforts in the creation of the Logo language, the National Center for Women & Information Technology awarded Solomon their Pioneer Award in 2016. She is now receiving much deserved recognition for her formative role in teaching young people to program.

Solomon clearly favored Piagetian learning styles, i.e., programming instruction through a student-driven, constructive process. Constructionists believe that people possess a range of theories about how the world works. Children's theories contrast sharply with adult theories. As children learn about the world, they build new intellectual structures using readily available materials, which they find in their own cultural setting. Children learn best when they are encouraged to draw on their own intuition and put to use what they already know as they develop new ideas.[42]

39. David Walden and Raymond Nickerson, eds., *A Culture of Innovation: Insider Accounts of Computing and Life at BBN* (East Sandwich, MA: Waterside Publishing, 2011), 281–532, here at 290.

40. Papert, *Mindstorms*, 212.

41. Cynthia Solomon, *Computer Environments for Children: A Reflection on Theories of Learning and Education* (Cambridge, MA: The MIT Press, 1986).

42. Solomon, *Computer Environments*, 103–104.

Papert and Solomon saw the computer as providing a useful context for new learning to take place. But for learning to work well, the computer's assets must be molded to the child's needs and meet them where they are. For this reason, the MIT Logo team pushed for newer and better computer hardware and software, and they wanted a programming language that would spur learning based on a child's personal knowledge.

The cultural environment where learning took place was also important to Papert and Solomon. In classrooms and learning centers, everyone should take on the dynamic roles of learner, teacher, theory builder, and theory tester. Learning should not be a passive process, in which the teacher (human or computer) simply pours knowledge into the student's head. As Solomon summarized, "The learning process becomes a shared responsibility among all participants."[43]

Papert also offered a vision for what he was looking for—a quasi-utopian environment that felt qualitatively different than a typical classroom or corporate training center in the 1960s:

> The [ideal] environment is designed to foster richer and deeper interactions than are commonly seen in schools today. Children create programs that produce pleasing graphics, funny pictures, sound effects, music, and computer jokes. They start interacting mathematically because the product of their mathematical work belongs to them and belongs to real life…
>
> Although the work at the computer is usually private it increases the children's desire for interaction. These children want to get together with others engaged in similar activities because they have a lot to talk about… By building Logo in such a way that structured thinking becomes powerful thinking, we convey a cognitive style, one aspect of which is to facilitate talking about the process of thinking.[44]

3.8 Logo as a Model for Code Nation

As we observed with FORTRAN learning, programming primers and other resources played an important part in the adoption and proliferation of new programming languages. This was also true for Logo, which was usually embedded in the school curriculum where it was implemented. A key component of this pedagogy was the belief that every child could learn to program, so the learning materials needed to be accessible for all learners, not just the most precocious.

The first version of Logo was introduced at the Hanscom Field School in Lincoln, Massachusetts in 1967, with support from the U.S. Office of Naval Research. During

43. Solomon, *Computer Environments*, 162.

44. Papert, *Mindstorms*, 180.

the 1968–1969 academic year, the National Science Foundation followed up with a year of teaching Logo-based math in elementary and middle school classrooms. The class materials and instructor's manual were designed and taught by Seymour Papert and Cynthia Solomon. Later in 1969, Logo was introduced to elementary school children at Emerson School in Newton, Massachusetts. These young programmers experimented with a variety of topics. They wrote Logo routines that translated English into "Pig Latin," devised "secret codes" (i.e., shifted the characters programmatically to make words appear garbled), and they created word games including trivia contests, word-reversing utilities, and string-processing routines.[45] A few years later, another Logo trial took place at Lincoln School in Brookline, Massachusetts. Again, the test subjects were 9 to 14 year olds.

The nationwide distribution of Logo teaching materials began when official textbooks were published and sold from coast to coast. One of the first was *Learning with Logo* (1983), written by Daniel Watt, a teacher and writer who taught elementary school for 7 years in Brookline and was invited to join the MIT Logo team in 1976.[46] Watt joined the group when the first microcomputers were making their appearance, and he was deeply influenced by Papert's educational philosophy. Between 1977 and 1981, Watt taught Logo in the Brookline schools, participating with a team of enthusiastic colleagues and volunteers. During those years, Watt worked closely with Brookline's School Superintendent, Bob Sperber, as well as other teachers who joined the effort to move Logo out of the "ivory tower" and into "real" classrooms.[47]

Learning with Logo is a spiral-bound workbook containing 14 chapters that introduce Logo concepts step by step. As Watt notes in the acknowledgments, many of the programs and exercises in the book originally came from his MIT colleagues. A few coding samples were also contributed by Brookline teachers and students at Lincoln School. Watt acknowledged the college textbook *Turtle Geometry* (1981) in his preface, which pioneered the teaching of college-level mathematics with Logo.[48] This tutorial was authored by MIT professors Harold ("Hal") Abelson and Andrea diSessa, and it offered a robust plan for teaching college-level math and geometry using computers. In short, Watt's *Learning with Logo* was a highly collaborative project, reflecting the Logo community's contributions and values.

More programming primers followed. As the Logo system made its way to new platforms, Watt's book was adapted from one computer platform to the next.

45. Walden and Nickerson, eds., *A Culture of Innovation*, 291.

46. Daniel Watt, *Learning with Logo* (New York: McGraw-Hill, 1983).

47. Daniel Watt, *Learning with Apple Logo* (New York: McGraw-Hill, 1984), FM.

48. Harold Abelson and Andrea diSessa, *Turtle Geometry: The Computer as a Medium for Exploring Mathematics* (Cambridge, MA: The MIT Press, 1981).

New editions included *Learning with Apple Logo* (1984), *Learning with IBM Logo* (1985), *Learning with Commodore Logo* (1985), and *Learning with Atari Logo* (1988), all published by McGraw-Hill. In terms of software sales, Apple Logo for the Apple II+ computer was generally recognized as the most popular implementation of PC-based Logo in the early to mid-1980s. This outcome makes sense given Apple's early commitment to the educational marketplace.

3.9 How successful was Logo?

In 1986, it was estimated that 40% of American school districts had acquired Logo and were using it to introduce programming fundamentals to students.[49] By that time, Logo had become more or less the standard learning language in elementary and middle schools, while BASIC and Pascal were widely adopted in high school and college contexts.

Looking back from her vantage point in the mid-1980s, Cynthia Solomon could admit that Logo as it had been taught since the late 1960s did not always achieve its goals. For example, in standardized tests the students who had been exposed to Logo did not always show significant differences in problem solving abilities compared to students who had not.[50]

The educational researchers Roy Pea and D. Midian Kurland also pointed out that in comprehensive studies, the results of Logo on student cognitive development were mostly anecdotal and inconclusive. Any positive results appeared to Pea and Kurland as being related to programming acumen alone (an important achievement). But programming in Logo did not seem to translate into higher mental functions.[51]

There were also those who criticized the constructivist approach altogether. Piaget, Papert, and Solomon believed that if students were given the cultural tools to learn with, they would create their own knowledge by self-guided immersive experiences (discovery learning), with a minimum of instruction or intervention. But scholars like Paul Kirschner, John Sweller, and Richard Clark argued that the constructivist approach only worked when learners had sufficiently high prior knowledge to provide their own "internal" guidance. Skilled teachers were more important than Papert realized, they argued. Only a proven pedagogy could truly

49. Solomon, *Computer Environments*, 132.

50. Solomon, *Computer Environments*, 128.

51. Roy D. Pea and D. Midian Kurland, "On the cognitive effects of learning computer programming," *New Ideas in Psychology* 2, no. 2 (1984): 137–168.

bolster mental processing, i.e., the cognitive architecture that accompanies and sustains deep learning.[52]

Criticism also came from unexpected quarters. Worried about the negative influence of a rapidly expanding computer culture in America, Theodore Roszak argued in *The Cult of Information* (1994) that Logo was well-intentioned but fell into the same trap that other computer education schemes did. The author of the celebrated *Making of a Counter Culture* (1969) even singled out Daniel Watt's *Learning with Logo* as an oppressive workbook that was overly concerned with the dynamics of *power* and *control*. (For an introduction to the argument of *Making of a Counter Culture*, see Chapter 2.) Roszak wrote the following about *Learning with Logo*:

> The phrase *powerful idea*—taken from Papert—appears as a little flag that punctuates the presentation in each chapter. But as with all computer exercises, the mastery comes through adapting to the machine's way of doing things. The same ambiguous relationship between power and dependence remains in Logo as in other computer curricula; the same illusion of control hovers over Papert's microworlds.[53]

Roszak was particularly unimpressed with the "art" that novice students could create with Logo commands, and he attacked the "poetry" that Watt said a programmer could create if they instructed the computer to pick words from a vocabulary list at random. Roszak believed all of these efforts lowered the bar for a child's education. He sensed a reductive rule at work: "if the computer cannot rise to the level of the subject, then lower the subject to the level of the computer."[54] For technologists who had come up in the 1960s and found inspiration in *Making of a Counter Culture*, Roszak's critiques of Logo and artificial intelligence must have seemed like a buzz kill. Unrepentant, Roszak concluded his critique: "Children are gaining their computer literacy at the risk of becoming cultural cripples."[55]

To the first of these objections, Solomon wrote that it was hard to determine the actual effect of Logo on children in a classroom, because Logo teachers were instructed in different ways and the Logo computer systems were inconsistently

52. For a summary of these critiques, see Paul A. Kirschner, John Sweller, and Richard E. Clark, "Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching," *Educational Psychologist* 41, no. 2 (2006): 75–86.

53. Theodore Roszak, *The Cult of Information: A Neo-Luddite Treatise on High-Tech, Artificial Intelligence, and the True Art of Thinking,* Second Edition (Berkeley: University of California Press, 1994), 76.

54. Roszak, *The Cult of Information*, 78.

55. Roszak, *The Cult of Information*, 82.

deployed. Admittedly, the Logo project had lofty goals, but they were being only partially implemented with technology that was still in-process. Moreover, the constructionist school believed that the context for human development was always a *culture* and never an isolated *technology*.[56] Logo programming would change hearts and minds, they argued, when the culture was fully computerized and attentive to new ways of learning. But it would likely take decades.

In 1997, Seymour Papert also looked back over a long career in AI research and agreed that his vision had only been partially achieved. The Logo project certainly enjoyed its successes over the past decades, but computer literacy was still in the early phases of a much longer cultural transformation. Papert offered the metaphor that if learning to program was like air travel or the cinematic arts, then the Logo project was like the early DC3 airplane or the first films that simply aimed a stationary camera at actors on the stage. In short, the technologies associated with computer programming were gaining momentum, but society's understanding of the medium hasn't evolved too far yet.[57] Papert also suggested that computational thinking was itself a *cultural attribute*. "The real problem is not whether Logo 'succeeded,' but understanding the growth of a computer learning culture in which Logo plays an important, but not determining, part."[58]

In 1983, computer literacy theorists Pea and Kurland wrote in a more optimistic way about the four major skill areas that should be assessed when educators needed to make a determination about how well a given course of programming instruction is going. These four skill areas included (1) understanding the programming problem; (2) designing or planning a programming solution; (3) writing the program code that implements the solution; and (4) comprehending the written program solution and program debugging.[59] The researchers agreed that in most contexts, Logo instruction as it existed in the 1980s attempted to cover all four skill areas, with varying levels of success. I would suggest that skill area (4) comprehending the written program solution and program debugging, is the area where Logo made its biggest strides, paving the way for programming to be seen as an interactive, experiential task that taught students about the nature of computers and how they can manage information.

56. Seymour Papert, "Computer criticism vs. technocentric thinking," in *Logo 85: Theoretical Papers* (Cambridge, MA: The MIT Press, 1991), 54.

57. Seymour Papert, "Educational Computing: How Are We Doing?" *T.H.E. (Technological Horizons in Education) Journal* (June 1997): 78–80.

58. Papert, "Educational Computing: How Are We Doing?" 80.

59. Pea and Kurland, "On the cognitive effects of learning computer programming," 149–150.

Learning to program has always been a challenging task, and the problem has been exacerbated in the modern era with new commercial programming systems arriving at ever-frequent intervals. This chapter's investigation of assembly language, FORTRAN, and Logo programming has revealed that modern approaches to software development are built on decades of thoughtful experimentation and research. As the computer industry moved from batch-processing computers to time-sharing systems in the 1960s, programming became a more interactive prospect, and high-level languages further opened the door to nonspecialists. Academic journals, programming primers, and new methods of classroom instruction provided Americans with many new opportunities to learn about programming. From these experiences, an educational movement was born that encouraged students and average citizens to try computer programming as a way to enhance their cognitive development. This led to the creation of the Logo programming language, which theorists like Seymour Papert and Cynthia Solomon hoped would be ideal to teach children spatial reasoning skills, mathematics, and problem solving. The learn-to-program movement gained momentum through these ideas, and in the coming decades a more comprehensive agenda would be constructed that attempted to integrate programming activities into the middle school, high school, and college curriculums.

Admittedly, America's computing infrastructure was not robust enough to manage the task of putting every child in front of a computer to learn. As we shall see in future chapters, however, the arrival of PCs made some believe that the goal was still possible, and they advocated for a new round of convivial tools that would bring the promise of computing to the masses. Chapters 4 and 5 explore how the learn-to-program movement continued under the banner of the BASIC programming language, a tool that went through rapid changes as the "Revolution" progressed.

4 Advocating Computer Literacy

"The Lord has given me the tongue of a teacher, that I may know how to sustain the weary with a word."

Hebrew Bible, Isaiah 50:4

"Do you occasionally make mistakes? We do, watch."

Bob Albrecht, *My Computer Likes Me When I Speak in BASIC* (1972)

In October 1972, Bob Albrecht, Mary Jo Albrecht, Jerry Brown, and LeRoy Finkel published the first issue of a new computing tabloid entitled the *People's Computer Company Newsletter*. Created in the offices of Dymax, a small publishing company organized under the auspices of the Portola Institute in Menlo Park, California, the newsletter would soon have a big impact on computer education. The issue's opening manifesto read

> Computers are mostly used against people instead of for people
> used to control people instead of to free them
> time to change all that –
> we need a...
> People's Computer Company.[1]

In popular histories of the "PC Revolution," the People's Computer Company (PCC) is credited with intensifying popular interest in computing in the years immediately prior to the commercial release of microcomputers. Most recently, Joy Lisi Rankin highlighted the formative influence of the PCC in her book *A People's History of Computing in the United States*, arguing that the organization was

1. *People's Computer Company Newsletter* 1, no. 1 (October 1972), 1. Marc LeBrun, Jane Wood, and Tom Albrecht are also credited as contributing art to the first issue.

central to the Bay Area home-computing endeavor.[2] The overlapping networks of the region's major technologists is indeed striking. Nearby the Portola Institute housed Stewart Brand's *Whole Earth Catalog*. Just down the road, Doug Engelbart was experimenting with computer interfaces at Stanford Research Institute. Also in the area, hardware enthusiasts were tinkering with circuits at the Homebrew Computer Club. The PPC somehow connected all of these groups—through pooled spaces, inexpensive hardware, potlucks, wine, and Greek dancing. But most significantly, they shared ideas.

For Bob Albrecht, the main idea was BASIC, a language that would become the popular voice of the learn-to-program movement as it transitioned into the microcomputer era. This chapter explores the promotion of computer programming and computer literacy in the 1970s and 1980s, and a fascinating cast of characters associated with BASIC in San Francisco, Minneapolis, Hanover, Berkeley, and Albuquerque. Later in the book, we'll return to the devoted users of assembly language, Pascal, Forth, and C/C++—powerful tools that were used to create commercial applications for CP/M, MS-DOS, Microsoft Windows, and the Apple Macintosh. But the computer literacy movement reached critical mass by way of BASIC in America, the dominant language of time-sharing teleprinters and the first microcomputers and PCs. The most interesting part of this story is not BASIC itself, but the teachers, students, and entrepreneurs who coded in it.

4.1 Robert Albrecht and the Popularization of the Movement

Robert L. Albrecht (1930–) started his computing career at Honeywell in Minneapolis in the 1950s, and later worked at Burroughs and Control Data Corporation in Colorado. While in Denver, he had a life-changing experience when he had the opportunity to teach FORTRAN to high school students. The class became popular and soon Albrecht was driving around the region teaching young people how to program computers. He also met education specialists who were studying innovative ways to teach mathematics in schools, including preparing teachers for the impending introduction of computers.[3]

2. Joy Lisi Rankin, *A People's History of Computing in the United States* (Cambridge, MA: Harvard University Press, 2018), 235.

3. No comprehensive biographies exist for Robert Albrecht, but his fascinating career has been touched on by numerous authors, including Rankin, *A People's History of Computing*, chapter 3; Michael Swaine and Paul Freiberger, *Fire in the Valley: The Birth and Death of the Personal Computer*, Third Edition (Dallas, TX: The Pragmatic Bookshelf, 2014), 155–165; Steven Levy, *Hackers: Heroes of the Computer Revolution*, Updated Edition (Sebastopol, CA: O'Reilly Media, 2010), 165–173, 194–199; John Markoff, *What the Dormouse Said: How the Sixties Counter-culture Shaped the Personal Computer Industry* (New York: Penguin, 2005), 181–185, 262–287; Bob Johnstone,

In 1964, Albrecht moved back to Minneapolis and quit his job with Control Data. He established a teaching connection with the University of Minnesota, and soon gained access to the school's time-sharing system, which contained a fascinating new programming language from Dartmouth College in Hanover, New Hampshire. The program was called BASIC, and the language was based on the work of John Kemeny and Thomas Kurtz in 1963 and 1964. BASIC was an acronym that stood for Beginner's All-purpose Symbolic Instruction Code, and the language was designed specifically for newcomers to programming.

Bob Albrecht loved coding in BASIC and he considered the language a major upgrade over FORTRAN, especially if your goal was teaching other people how to write instructions for computers. The Minnesota implementation of BASIC was interactive (not slowed down by the delays of batch-processing), and it offered English-like statements that students could master quickly. Kemeny and Kurtz designed the language around a new time-sharing system, one of the first in America that was created in partnership with scientists from MIT.[4] This meant that programmers could use teletype machines or consoles to type their code directly into the computer. From the user's point of view, there was now a two-way flow of information between the system and the user when the program ran. BASIC's INPUT statement is one example of how this interactivity was deployed. INPUT allowed users to enter information into a program while the code is executing. When the statement is processed, the terminal session pauses and waits patiently for the user to supply the requested information. After receiving the input from the user, the program continues its processing and (if the program is skillfully designed) it can make use of this information through variables and other mechanisms. This type of interactivity—now a common feature of programming systems—was exciting and new for students and their instructors, offering programmers an intimate connection with their computers.[5]

Bob Albrecht's growing passion for BASIC can be gauged by his regular use of the acronym SHAFT (Society to Help Abolish FORTRAN Teaching), a slogan he put on buttons and business cards in the mid-1960s to influential policy makers.

Never Mind the Laptops: Kids, Computers, and the Transformation of Learning (Lincoln, NE: iUniverse, Inc., 2003), 65–69; and Steve Lohr, *Go To: The Story of the Math Majors, Bridge Players, Engineers, Chess Wizards, Maverick Scientists, and Iconoclasts—The Programmers Who Created the Software* Revolution (New York: Basic Books, 2001), 87–89.

4. For an excellent history of the creation of time-sharing BASIC at Dartmouth College, see Rankin, *A People's History of Computing in the United States*, 12–37.

5. The INPUT statement was introduced in the Third Edition of Dartmouth BASIC in 1966. See Thomas E. Kurtz, "BASIC session," in *History of Programming Languages*, ed. Richard L. Wexelblat (New York: Academic Press, 1981), 515–549, here at 527.

Albrecht's goal was to convince educators and politicians that BASIC was superior to FORTRAN when the objective was teaching people the fundamentals of computer programming. Gradually, this advocacy paid off. In Minnesota, a national mathematics education council met in subcommittee and agreed that BASIC should be selected as the primary teaching language in the districts under their control.[6] Although Minnesota might seem like the periphery in a story about early computing standards, this supposition is far from accurate. The council's decision exercised an important influence over computer-based education movements throughout the 1960s and 1970s.[7]

Following the policy shift, Addison-Wesley hired Albrecht and a few colleagues to write a book entitled *Computer Methods in Mathematics* that would elevate BASIC instruction over other languages for teaching math. The guidebook was among the first to teach teachers how to present mathematics instruction using BASIC.[8] (The book also included a short section on using FORTRAN, mostly to mollify stalwarts.) After beginning the project, Albrecht moved to California to be closer to his publisher—and to make a fresh start. He arrived in the bustling San Francisco Bay Area in early 1966 and immediately established contacts in the local hotbeds of publishing, art, education, and computing. Albrecht's charisma led to a string of entrepreneurial startups. In the publishing realm, Albrecht co-founded the Portola Institute, the PPC, and Dymax publishing. The PCC name was inspired by the Janis Joplin band Big Brother and the Holding Company. In reality, neither Big Brother and the Holding Company nor the PCC functioned much like a traditional "company" at the time. Both groups were experimenting with music, counterculture politics, and consciousness-raising activities.

In his spare time, Albrecht drove around with LeRoy Finkel and other new friends in a red VW bus, trying to teach young people in San Francisco about BASIC. They brought along computers, calculators, books, games, manuals, and anything else that they could find to spread the word about coding. In slightly more formal settings, they also offered courses to teachers and students on University of California branch campuses and in what was called the Midpeninsula Free

6. This anecdote comes from an interview with Albrecht's colleague Dale LaFrenz. Apparently, Albrecht personally lobbied the Computer Oriented Mathematics Committee, a subgroup of the National Council of Teachers of Mathematics. For the context of the decision, see "Interview with Dale LaFrenz," conducted April 13, 1995, Charles Babbage Institute Archives, University of Minnesota.

7. Rankin, *A People's History of Computing in the United States*, 139–165.

8. See Robert L. Albrecht, Eric Lindberg, and Walter Mara, *Computer Methods in Mathematics* (Menlo Park, CA: Addison-Wesley Educational Publishers, Inc., 1969).

University (MFU), which operated primarily between 1966 and 1971 on the San Francisco Midpeninsula.

The MFU teaching experience was especially congenial to its times. MFU was one of the larger "free" universities that settled on college campuses, private homes, and storefronts in the late 1960s in response to the Free Speech Movement at Berkeley. MFU offered hundreds of courses each quarter for a nominal membership fee of $10. The events were publicized in illustrated catalogs and widely distributed. Albrecht and Finkel taught MFU computer courses with a countercultural sensibility, working to disassociate themselves from the negative stereotypes that computers had for some hippies. In particular, many in the Free Speech Movement worried that computing seemed "too corporate," or that computers were closely allied with the military-industrial complex, which many were actively protesting against. Albrecht recalled his initial strategy: "We were covert. Unintentionally, we were taking the long-term view, encouraging anyone who wanted to use computers, writing books that people could learn to program from, setting up places where people could play with computers and have fun."[9]

As Figure 4.1 illustrates, the learn-to-program movement had picked up a new language and progressed to a new stage.

4.2 I Speak BASIC

Over his career, Bob Albrecht would author or co-author over 30 computer books, mostly about BASIC programming and mathematics. Through Dymax, Dilithium Press, Addison-Wesley, John Wiley, and other publishers, Albrecht drew in thousands of new computer users and taught them to love BASIC. He introduced readers to the fundamentals of computer literacy through light-hearted prose, interesting math problems, computer games, and conference presentations. (See Figure 4.2 for Albrecht presenting at the West Coast Computer Faire.) Most of Albrecht's programming primers were large-format workbooks, relatively short in length but packed with original artwork—and on occasion, encouragement from mythological characters. In fact, dragons and mythology were favorite subjects for Albrecht. In later years, he occasionally wrote under the penname George Firedrake, a dragon alter-ego that he developed after taking a free-university course on self-hypnosis at Foothill College in Los Altos.

Albrecht's whimsical texts were a far cry from the academic approach that most professors were taking when they introduced programming and the fundamentals of computer science. Although a few programming primers had appeared for the BASIC language, they tended to resemble the engineering manuals of an earlier era.

9. Bob Albrecht, quoted in Levy, *Hackers: Heroes of the Computer Revolution*, 169.

Figure 4.1 **Illustrations from the *People's Computer Company Newsletter* were often linked to imagery from the Free Speech Movement to popularize the invitation to learn programming. This graphic, repeated in several PCC newsletters, first appeared in October 1972. It presents a diverse group ready to code: young and old, black and white, female and male.** (Courtesy of the Computer History Museum)

These books included Kemeny and Kurtz's *BASIC Programming* (1967, 1971) and James S. Coan's *Basic BASIC* (1970, 1978).[10] I would also be remiss if I don't add to this list the first BASIC programming textbook that I used in a college course, the time-honored *BASIC: An Introduction to Computer Programming*, by Robert Bent and George Sethares (1978, 1982).[11] All of these texts introduced essential language concepts and syntax, but they read like language specifications with little in the way of real-world examples. In other words, they were separated from the lived experience of young people. Albrecht wanted to change this, so he emphasized counterculture values, humorous anecdotes, and what he considered to be interesting about

10. John G. Kemeny and Thomas E. Kurtz, *BASIC Programming* (New York: John Wiley, 1967; Second Edition, 1971); James S. Coan, *Basic BASIC*: *An Introduction to Computer Programming in BASIC Language* (New York: Hayden, 1970; Second Edition, 1978).

11. Robert J. Bent and George C. Sethares, *BASIC: An Introduction to Computer Programming* (Monterey, CA: Brooks/Cole Publishing, 1978 and 1982). Characteristically, my BASIC programming course was a semester long elective which did not count for credit in my university's computer science major. I took it anyway.

Figure 4.2 **Bob Albrecht speaking at the March 1980 West Coast Computer Faire in the Civic Center, San Francisco.** (Photograph by Jim Warren; image courtesy of Jim Warren and the Computer History Museum)

the world. Albrecht also had business savvy. To multiply his efforts, he formed partnerships with several technical writers and teachers in the Bay Area, and the group published dozens of books together. These skilled collaborators included LeRoy Finkel, Jerry Brown, Don Inman, Ramon Zamora, Greg Stafford, and (when Visual Basic arrived) Bob's own son, Karl Albrecht.

All the evidence suggests that Bob Albrecht and his team made BASIC programming fun and memorable for their intended audiences. Albrecht's first major book, *My Computer Likes Me When I Speak in BASIC* (1972), set the tone for the group's adventures. (See Figure 4.3.) When describing the teletype equipment that he used to enter and run BASIC programs, Albrecht explained: "Teletypewriters are the Volkswagens of computer terminals... rugged, dependable, inexpensive, ugly and noisy!"[12] After this introduction to equipment, Albrecht proceeded to step people through building interesting, interactive BASIC programs that were an appropriate match for the era's mainframe and minicomputer systems. Importantly, he

12. Bob Albrecht, *My Computer Likes Me When I Speak in BASIC* (Portland, OR: Dilithium Press, 1972), 1.

Figure 4.3 **The cover of *My Computer Likes Me When I Speak in BASIC* (1972), by Bob Albrecht.** (Courtesy The Internet Archive, San Francisco)

recognized that people were *still learning* about computers and their capabilities. This was an awareness that the best computer book authors have always seemed to possess.

A timely issue that Albrecht investigated in his first book was global population growth. Although the problem might seem obscure today, population growth was an important political and environmental concern in the 1960s and 1970s. Popular publications such as Stewart Brand's *Whole Earth Catalog* and Ted Nelson's *Computer Lib/Dream Machines* made numerous references to global population growth and the ethical and political issues that it raised. Albrecht's concern with the topic reminds us that even in computer books, popular culture influences pedagogy. It is likely that Albrecht's readers were intrigued, as well.

A recent contributor to the debate was the biologist Paul R. Ehrlich, author of the influential book *The Population Bomb* (1968). Ehrlich warned readers that the earth's human population would soon increase to the point of food shortages, system collapse, and mass starvation. Stewart Brand had studied biology with Ehrlich

at Stanford in the late 1950s, and Brand helped introduce Erlich's ideas to a broader audience in the San Francisco Bay Area.[13] Bob Albrecht was one of the people that absorbed this message.

Although it may sound like a complex subject for new programmers to consider, Albrecht recognized that human growth rates were fascinating, and they could easily be adapted to computational logic. Albrecht's section "Too Many People" near the beginning of *My Computer Likes Me When I Speak in BASIC* takes as its starting point the world's population of 3.6 billion people in 1970. (Those were the days, right?) Using the standard growth rate accepted in that era (the earth's human population doubling every 35 years), Albrecht showed readers how to write a short program that would compute the earth's population in 2250 (i.e., 280 years into his model reader's future). With the growth rate that Albrecht proposed, the computation worked out to an astonishing 921.6 billion people by the year 2250. No wonder they were so worried.

Naturally, there are some physical limits to population expansion that are not considered in this example. For one thing, as the earth fills up and resources are depleted, the growth rate will presumably slow. But Albrecht's point was not to dive too deeply into population growth theory—he wanted to inspire readers to write programs that might address interesting contemporary issues. Albrecht simplified this first example by picking an end year and a growth rate that could be expressed by a power of 2. The solution he presented can be calculated using the following formula:

Human population in the year 2250 $= 3{,}600{,}000{,}000 \times 2^8$

When translated into BASIC code, Albrecht's formula looked like this:

```
10 PRINT 3.6E9*2*2*2*2*2*2*2*2
99 END
RUN
```

This is a complete BASIC program, which Albrecht printed carefully in his book for readers to type in via their teletype terminals. The first two lines begin with line numbers (10 and 99), *labels* that were required to distinguish each line in early versions of BASIC. (Different line numbers could be used, but it was typical to use 10, 20, 30, and so on to allow room for later insertions.) The third line is simply a RUN command, which directs the interpreter to start running (or executing) the program. When the program starts, it printed the result of the calculation on the teletype:

```
9.216000E+11
```

In plain English, this numeric expression means that in the year 2250 the earth's population will be (theoretically) 921,600,000,000 people, which we might simplify

13. Andrew G. Kirk, *Counterculture Green: The Whole Earth Catalog and American Environmentalism* (Lawrence, KS: University Press of Kansas, 2007), 33–36.

using the number 921.6 billion.[14] Albrecht then wrote about the exponential part of the result, so that readers could understand how BASIC handled such numbers. Later in the book, he encouraged readers to enter numbers using this format, which the computer could understand and which (theoretically) saved the programmer time and space.

I have taken the time to review Albrecht's sample problem in detail because it is useful to understanding his method when introducing computation thinking with BASIC. A close reading of his primer reveals more than a hint of social commentary which accompanies his exercises. It continued in at least another location, where he wrote (in a second typeface), "Too many people!" next to the 921.6 billion figure. He also provided readers with information that they could use to learn more about population growth, including writing to the Population Reference Bureau in Washington, D.C.[15] (The bureau's address is printed on a sample envelope next to the text—just what an experienced activist would do when providing instructions to the members of a social movement.)

Albrecht's *My Computer Likes Me When I Speak in BASIC* was financially successful, selling over 250,000 copies. Beyond mere revenues, it indicates that thousands of readers were learning BASIC and connecting the language to computer instruction and countercultural ideas. The text also captures a moment when most novices were learning BASIC on time-sharing systems, before the release of the first microcomputers. Along with Daniel McCracken and David Ahl, Bob Albrecht became one of the learn-to-program movement's early champions and recognizable leaders.

4.3 The B. F. Skinner Approach

As the 1970s took shape, BASIC programming remained a priority for the Dymax/PCC team. When the first microcomputers arrived, the publishing group revised several titles for the new devices, including a book known simply as *BASIC*. Albrecht, Finkel, and Brown emphasized the new computer's novelty in a foreword to the 1978 edition (italics mine):

> Since the appearance of BASIC... the field of computer science and the availability of computers *to all people* (non-professional computer users) has grown by leaps and bounds. Especially noteworthy *is the appearance of the so-called personal or home computer*. Integrated circuit technology has now provided us with computers far less expensive than ever before, yet with the same computing abilities as systems costing many times more. This means it will

14. Albrecht, *My Computer Likes Me*, 12.

15. Albrecht, *My Computer Likes Me*, 23.

be easier for you, the beginner, to get "hands-on" computer programming practice in the BASIC language.[16]

In addition to a flourish of marketing, there are echoes here of Ivan Illich, Lee Felsenstein, and Ted Nelson—the prophets of convivial tools "for the people." This primer is also noteworthy for a pedagogical reason; the book is a prime example of an approach to computers utilizing the techniques of behaviorism and operant conditioning from the field of developmental psychology. *BASIC* was published by John Wiley & Sons, a venerable technical publisher that began its relationship with Bob Albrecht through Commissioning Editor Judie Wilson in the early 1970s.[17] At that time, Wiley's editorial team was invested in an approach to learning popularized by B. F. Skinner (1904–1990) and a team of behavioral psychologists. The system was more recently promoted by Susan Markle (1928–2008), a psychologist and instructional designer who published the treatise *Good Frames and Bad* (1969).[18]

Wiley & Sons believed that novice computer programmers could benefit from operant conditioning because programming syntax was governed by well-defined rules. Following the method, the Wiley editors organized learning exercises into question/answer (stimulus/response) "frames," which exposed learners to new concepts through controlled steps that could be reinforced. Each frame began with a short conceptual description and then presented a simple task to the user that was learned through short exercises.

A single frame presented only one small part of the overall programming skill that was to be introduced. However, when looked at collectively, the frames were designed to generate a comprehensive proficiency in the skill being taught. The key was providing immediate feedback that would shape and reinforce positive behavior. Self-test questions were also included in the books, and the solutions were shown directly below the questions. (Readers were encouraged to hide the solutions with a note card until they were ready to check their work.) Computer-based training courses that followed this method would be able to do this part automatically.

I find this approach to teaching fascinating, because it had a long influence in the fields of instructional design and computer book publishing, some of which I witnessed firsthand at Microsoft Press. However, Wiley's *BASIC* book seemed like

16. Robert L. Albrecht, LeRoy Finkel, and Jerald R. Brown, *BASIC*, Second Edition (New York: John Wiley & Sons, 1978), v.

17. "Interview with Bob Albrecht," conducted by Jon Cappetta, April 7, 2015, History of Computing in Learning and Education.

18. Susan Meyer Markle, *Good Frames and Bad: A Grammar of Frame Writing*, Second Edition (New York: John Wiley & Sons, 1969). For additional background on the method, see B.F. Skinner, *The Technology of Teaching* (New York: Appleton-Century-Crofts, 1968).

a bit of a departure for Bob Albrecht and his colleagues. The group's sense of humor lost some of its sparkle in the new format, and the illustrations were removed, along with some of the freshness of their approach. However, Wiley was confident in the new method, and they filled *BASIC* with scores of numbered exercises that stepped readers through fundamental programming concepts and conditioning tasks. Some of these could be very lengthy. For example, the book's opening section, "Warming Up," stretched for 42 pages, presenting task after task with little explanatory text.

Not everyone agreed with the operant conditioning method. As I discussed in Chapter 3, Seymour Papert and Cynthia Solomon were deeply suspicious of rote learning, and they encouraged teachers and students to approach cognitive development through immersive, interactive learning systems that were fun. In fact, most of Albrecht's computer books followed a path similar to what Papert and Solomon had advised. Although the learn-to-program did brush up against these behaviorist approaches to computer literacy, most of the movement's authors and educators employed fun, hands-on learning experiences that were meant to be immersive when teaching programming. The arrival of personal computers (PCs) only enhanced this mode of exploration.

4.4 Hold Me Closer Tiny BASIC

Interestingly, Bob Albrecht had another important legacy in early programming communities—he supported the free distribution of BASIC on the first PCs, encouraging the language's success in the rapidly rising platform. Although this influence has been noted by other historians of technology, it bears repeating that BASIC moved very seamlessly from mainframe to minicomputer to PC contexts, and Bob Albrecht played a crucial role in this technology transference.[19]

In early 1975, Bill Gates and Paul Allen licensed their version of the BASIC interpreter to MITS in Albuquerque for use on the Altair 8800 microcomputer. As there was little else in the way of software for the Altair system, hobbyists were forced to write BASIC programs with the Gates–Allen–Davidoff version (Altair BASIC) when they wanted to make use of the new device. As newer microcomputer systems came out, the nascent "Micro-soft" team converted their BASIC interpreter so that it would run on the new computers, licensing the software to manufacturers who wanted to use it. They typically sought a royalty for each copy of BASIC that was sold.

19. For another account of this transition, see Rankin's, *A People's History of Computing in the United States*, 66–105.

From a user's point of view, the only problem with this scenario was that it added (potentially) significant costs to the new system. For example, Altair BASIC was originally priced at $150 for the 4K interpreter program—over a third of the cost of the $439 Altair 8800 microcomputer kit.

Recognizing the cost to users and finding himself in a position of influence in the community, Albrecht devised a creative solution. He contacted Dennis Allison, a computer science instructor at Stanford University, and he asked Allison to write a compact (tiny) version of BASIC that the PCC could publish in their newsletter for free. Presumably, the free interpreter would encourage people to code in BASIC, the language that Albrecht loved. Free code was also in the spirit of the group's vision for "convivial tools," and they hoped that widespread distribution of the enabling software would encourage hobbyists to experiment further with microcomputers.

The "free BASIC group" went to work. In 1975, the first three issues of the *PCC Newsletter* included articles written by Allison calling for "tiny" (2–3 kilobyte [KB]) versions of BASIC that might fit easily into a typical microcomputer's memory allocation. Allison offered technical specifications for the interpreter, and he named the tool *Tiny BASIC*. In the final article in the series, Allison asked readers to mail in their own ideas for the BASIC interpreter.

The response to the Tiny BASIC initiative was overwhelming. Because there was so much interest in a scaled-down programming language for the new devices, the PCC team decided to create a separate magazine to share the source code for BASIC and publisher reader feedback about the experience. At a loss for a name for the publication, the PCC editors finally came upon the title *Dr. Dobb's Journal of Tiny BASIC Calisthenics & Orthodontia*. Later, the PCC hired computing veteran Jim Warren to run the magazine, and the publication's name was simplified to *Dr. Dobb's Journal*. This venerable magazine would become a mainstay of the PC programming world, reaching tens of thousands of active software developers throughout the 1980s and 1990s.[20] I have much more to say about Jim Warren, but I'll leave it to Chapter 11, which introduces important trade shows in the PC industry, including the West Coast Computer Faire.

By the middle of 1976, there were many low-cost or free implementations of Tiny BASIC in the microcomputer community. The Micro-soft version was still the most popular, but the PCC and *Dr. Dobb's Journal* had helped to create a broad-based movement around BASIC, and the "underground" nature of the tool encouraged hobbyists to meet and exchange software. We can track the movement's

20. Jim Warren describes these early events in a commemorative edition of *Dr. Dobb's Journal*, where I have gathered the information. See Jim Warren, "We, The People, In the Information Age: Early times in Silicon Valley," *Dr. Dobb's Journal*, no. 172 (January 1991): 96D–96H.

expansion via local computing circles by examining the May 1976 issue of the *PCC Newsletter*, which contains a list of 104 active computer clubs in the U.S. Most of these were early adopters of microcomputers and the BASIC programming tools mentioned in the *PCC Newsletter*. In the same issue, the editors published a list of retail stores where new computer products could be purchased. In 1976, there were already 47 active businesses that could be identified, supporting a range of microcomputer kits and products.[21] Collectively, the clubs and stores allowed programmers, engineers, and hobbyists to meet, find hardware, learn new skills, trade games, buy books, collaborate on projects, and develop shared values. This face-to-face contact was crucial for the expansion of the learn-to-program movement and its grassroots ideals.

4.5 Arthur Luehrmann and the Computer Literacy Debate

BASIC's popularity on microcomputers and time-sharing systems insured its relevance for publishers and entrepreneurs, but the language's success also propelled the product into initiatives related to educational policy and literacy agendas around the country. The next phase of this movement took place at Dartmouth College in New England. Among the important supporters of BASIC at Dartmouth was Arthur W. Luehrmann, Jr. (1931–), a physics professor who became a tireless advocate for the learn-to-program movement in the 1960s and 1970s. Luehrmann was a colleague of Kemeny and Kurtz who helped to institutionalize BASIC's use at the college's Kiewit Computation Center. When the language attracted national attention, the Dartmouth group took their show on the road to demonstrate the value of programming in a time-sharing context. Following their lead, the administrators of the Dartmouth system provided access to BASIC for regional high schools and universities that otherwise had little contact with computers.

Arthur Luehrmann had a Ph.D. in Physics from the University of Chicago and he was an assistant editor of the *American Journal of Physics*, a publication of the American Association of Physics Teachers. Despite the draw of his discipline, the physicist also spent much of his time tinkering with computing hardware, as many scientists did when they had their first taste of computing. In 1968, Luehrmann devised a pioneering approach to printing graphics from a BASIC program using an X–Y plotter, a peripheral that he connected, via another device, to the Dartmouth time sharing system. Luehrmann's technique was essentially a *hack* (or a workaround). He connected the plotter to a teletype printer, and then used BASIC routines with embedded escape sequences to send output to the plotter via the

21. See "Clubs," *People's Computer Company Newsletter* 4, no. 6 (May 1976): 28–29.

printer. Luehrmann wrote a pamphlet about the clever technique, which is still available for review in the Dartmouth College library.[22] The results encouraged faculty and students to make further experiments with graphics at their school. The program also drew the attention of Tektronix, Inc., which loaned Luehrmann's team several graphics terminals to enhance their work.[23] Luehrmann collaborated with Tektronix throughout the 1970s, using their influence to lobby for the inclusion of graphics commands in the ANSI standard version of BASIC.

From this point on, Luehrmann regularly emphasized computer-based education in his work. In 1971, the Dartmouth team started Project COMPUTe, a 3-year program funded by the National Science Foundation to create course materials supporting the integration of computers in college classrooms. Thomas Kurtz was the project principal and Luehrmann was appointed the project director. In 1972, Luehrmann wrote a seminal article that was among the first to propose a working definition for "computer literacy" or, as Luehrmann called it, *computing literacy*. Luehrmann began his article with a parable that emphasized the importance of reading and writing for all well-educated students. He then called for a similar mastery over the computer's impressive capabilities:

> If the computer is so powerful a resource that it can be programmed to simulate the instructional process, shouldn't we be teaching our students mastery of this powerful intellectual tool? Is it enough that a student be the subject of computer administered instruction—the end-user of a new technology? Or should his education also include learning to use the computer (1) to get information in the social sciences from a large data-base inquiry system, or (2) to simulate an ecological system, or (3) to solve problems by using algorithms, or (4) to acquire laboratory data and analyze it, or (5) to represent textual information for editing and analysis, or (6) to represent musical information for analysis, or (7) to create and process graphical information?

22. Arthur W. Luehrmann, *Use of the Time Share Peripherals plotter in the Dartmouth GE-635 TSS* (Hanover, NH: Kiewit Computation Center, 1968). The Dartmouth copy is the only known printing of this booklet that survives. For a shorter description of Luehrmann's technique, see John G. Kemeny and Thomas E. Kurtz, *Back to BASIC: The History, Corruption, and Future of the Language* (Reading, MA: Addison-Wesley, 1985), 40–43.

23. In 1969, Tektronix gave Dartmouth a Model 4002 graphics terminal, which an advertisement describes as "the first graphic terminal priced under $10,000." The Tektronix ad includes a photo of Luehrmann working with students and information about the partnership with Dartmouth. See "Back Matter," *Scientific American* 234, no. 3 (1976). http://www.jstor.org/stable/24950318. Accessed August 20, 2019.

> These uses of computers in education cause students to become masters of computing, not merely its subjects.[24]

Notice that the ability to *write computer programs* is a prerequisite for several of the items on Luehrmann's list, including formulating database queries, simulating an ecological system, and devising algorithms. Luehrmann also emphasized the interdisciplinary nature of computing literacy, highlighting the contributions of the liberal arts, the social sciences, the natural sciences, and music. Elsewhere, Luehrmann recommended that training in computer programming should begin as early as the seventh grade to prepare students for computing literacy subjects at the high school and college levels.

Luehrmann was most familiar with BASIC, so he recommended the language for use in the classroom. We know from his teaching notes that he often required first-year Physics students to write short programs to master the scientific principles that they were studying. For example, Dartmouth students were required to write 10- to 20-line BASIC programs that calculated and plotted orbits around planets using Newton's Laws of Motion.[25]

Luehrmann was not zealously committed to BASIC's original syntax. When the language came under attack in the late 1960s and early 1970s for its lack of procedural features, Luehrmann advocated for changes to BASIC and the introduction of structured programming techniques. He also appreciated Pascal and later co-wrote a book about its use.[26] Luehrmann claimed that he was not just teaching syntax and keywords, "but conceptual language where the student is thinking in terms of procedures and sub-procedures and loops and branches and all of these complex ideas that are very powerful."[27] To the best of his ability, he emphasized inquiry, critical thinking, and experimentation in his advocacy for computing literacy. At Dartmouth, he couched this using the cognitive terminology of his era: "we're taking students out of their passive receptor situation and putting them into the active inquiry role of a researcher."[28] (At the Kiewit Computation Center, the process could be quite lively, as the 1969 photo in Figure 4.4 suggests.) It was a strategy that

24. Arthur W. Luehrmann, "Should the computer teach the student, or vice versa?" *Proceedings of the Spring Joint Computer Conference* (Montvale, NJ: AFIPS Press, 1972): 407–410, here at 410.

25. Arthur W. Luehrmann, "'Should the computer teach the student…'—30 years later," *Contemporary Issues in Technology and Teacher Education* 2, no. 3 (2002): 397–400.

26. Arthur Luehrmann and Herbert D. Peckham, *Apple PASCAL: A Hands-on Approach. Programming Languages Series* (New York: McGraw-Hill, 1981).

27. Luehrmann quoted in "Back Matter," *Scientific American* 234, no. 3 (1976). http://www.jstor.org/stable/24950318. Accessed August 20, 2019.

28. "Back Matter," *Scientific American* 234, no. 3 (1976).

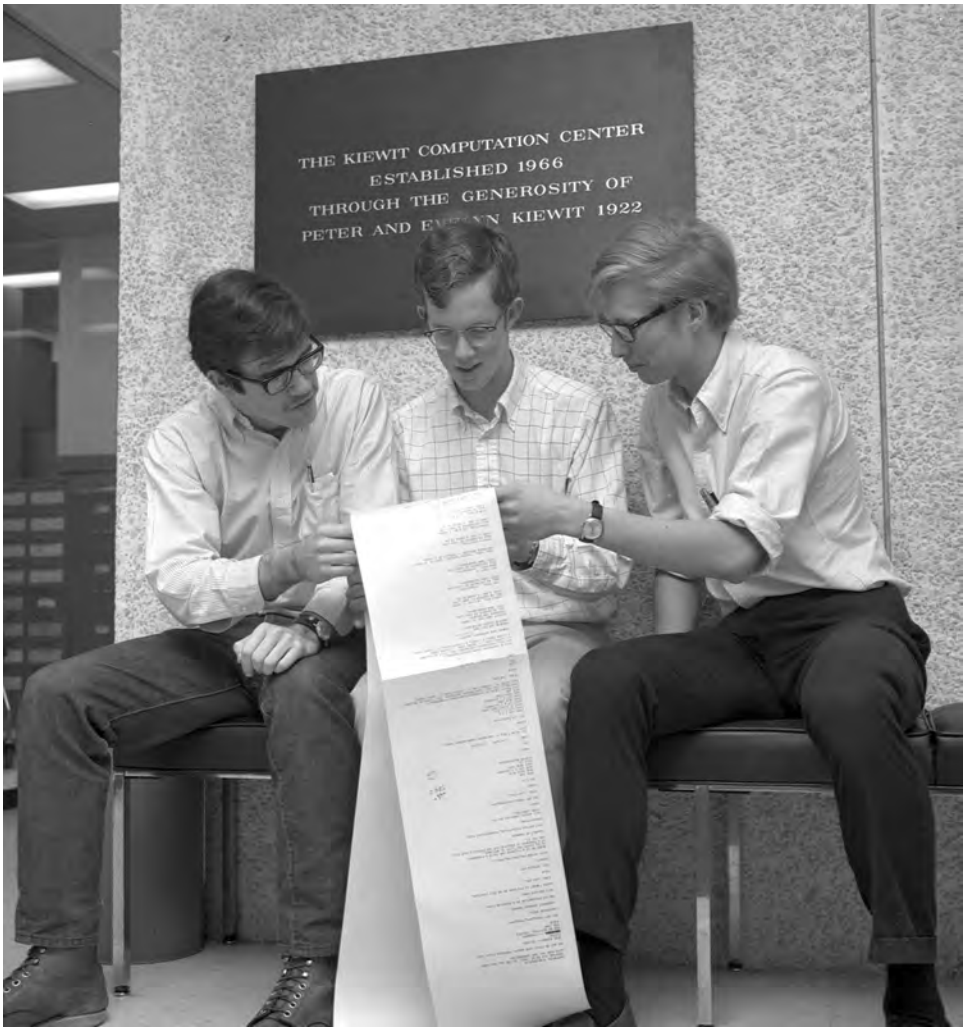


Figure 4.4 **Dartmouth students study a program listing outside the Kiewit Computation Center in 1969.** (Photo by Adrian N. Bouchard. Courtesy of Dartmouth College Library)

computer educators across the country were using as they introduced BASIC to students in schools and community-centered programs.

Like Seymour Papert and Cynthia Solomon, Arthur Luehrmann also encountered critics when he shared his vision for broad-based computer literacy programs that involved programming. In 1980, Luehrmann came under fire from opponents who believed that his definition of computing literacy was too focused on software development and overly dependent on BASIC. As PCs became more

affordable and found their way into schools, school administrators needed clarification about which computer subjects should be taught and how they should be introduced. Few observers disputed that computer literacy was important, but the question hinged on the scope of coverage and the subjects that should receive emphasis. Did computer literacy mean *programming*, or could the term be understood more generally, incorporating social, ethical, and business topics that related in interesting ways to computing? As useful software applications became available, should students spend more time learning spreadsheet and word processing skills, and less time writing computer programs? Or was there some appropriate mixture that wouldn't choke the existing curriculum and squeeze out traditional subjects like reading, mathematics, music, and art?

In the journal *The Mathematics Teacher*, four prominent researchers from the field of education weighed in on the issue—against Luehrmann's proposals. David Johnson, Ronald Anderson, Thomas Hansen, and Daniel Klassen argued that a much more comprehensive definition of computer literacy should be adopted in K-12 schools than Luehrmann proposed. Using research funded by the Minnesota Educational Computing Consortium and the National Science Foundation, they proposed a list of 63 comprehensive learning objectives that should be used to define and assess computer literacy across the country.[29] The researchers listed cognitive and affective learning objectives that might include programming skills but also emphasized a host of social and cultural aptitudes related to computers. These objectives included identifying the basic operation of a computer system; comparing computers and their capabilities to the human brain; determining how computers can assist consumers; recognizing that computers can be used to commit serious crimes; recognizing that computerization both increases and decreases employment opportunities in the economy; instilling confidence in students so that they felt they could use and control the new machines; and much more. In short, they recommended that computer literacy expand its scope to become *computer awareness*. The team called for further study and assessment, leaving open the possibility that additional items could be added to the list. They also asked local educators to build a curriculum that would appropriately prioritize and integrate their list of priorities.

The group's recommendations generated a firestorm among teachers, administrators, and parents who were all seeking the best ways to integrate computers into the K-12 curriculum. At the 1980 annual meeting of the National Council of Teachers of Mathematics, the most popular session was the panel in which Daniel

29. David C. Johnson, Ronald E. Anderson, Thomas P. Hansen, and Daniel L. Klassen, "Computer literacy—What is it?" *The Mathematics Teacher* 73, no. 2 (February 1980): 91–96, here at 96.

Klassen presented his group's recommendations to the organization. At the center of the debate was a concern about the tremendous costs that computing-centered initiatives would bring to districts.

To bring some clarification to the issues, *The Mathematics Teacher* invited letters from Luehrmann and the team of Minnesota researchers to clarify their positions and propose possible solutions. The letters were published in the December 1981 issue of the journal, and they have been reissued over the years as districts continue to debate the importance of literacy and programming initiatives in the educational curriculum. The debate had a significant impact on the learn-to-program movement, and we can hear its echo in modern discussions about computational literacy and science, technology, engineering and mathematics (STEM) funding for districts and students.

Luehrmann's letter argued that computer literacy implies gaining fluency over the structures and processes of a computer so that the devices can be made to do useful work. In 1981, there were few comprehensive software packages for PCs, so accomplishing useful work for Luehrmann meant writing your own programs in an accessible language. (Although he preferred structured BASIC, Luehrmann didn't insist on a particular language in his response.) The Dartmouth professor wrote that computer literacy meant the ability to *do computing* as one demonstrates other literacies by *reading* books, *writing* letters, and *solving* math problems. (See Figure 4.5.) He clarified his position with a simple analogy (the italics are Luehrmann's):

> Literacy in a language means the ability to read and write, that is, to *do* something with language, not merely to *recognize* that language is composed of words, to *identify* a letter of the alphabet, or to be *aware* of the pervasive role of language in society. Literacy in mathematics means the ability to add numbers, solve equations, and so on—to *do* mathematics, not merely to *recognize* that numbers are written as sets of digits or to *identify* a fraction by its appearance or to be *aware* of the vocational advantages of being able to do mathematics.
>
> By analogy, computer literacy must also mean the ability to *do* computing, and not merely to recognize, identify, or be aware of alleged facts about computing.[30]

Luehrmann drew attention to the very long list of learning objectives proposed by the Minnesota researchers. Reacting in dismay to the list's length, he

30. Arthur Luehrmann, "Computer literacy – What should it be?" *The Mathematics Teacher* 74, no. 9 (December 1981): 682–686.

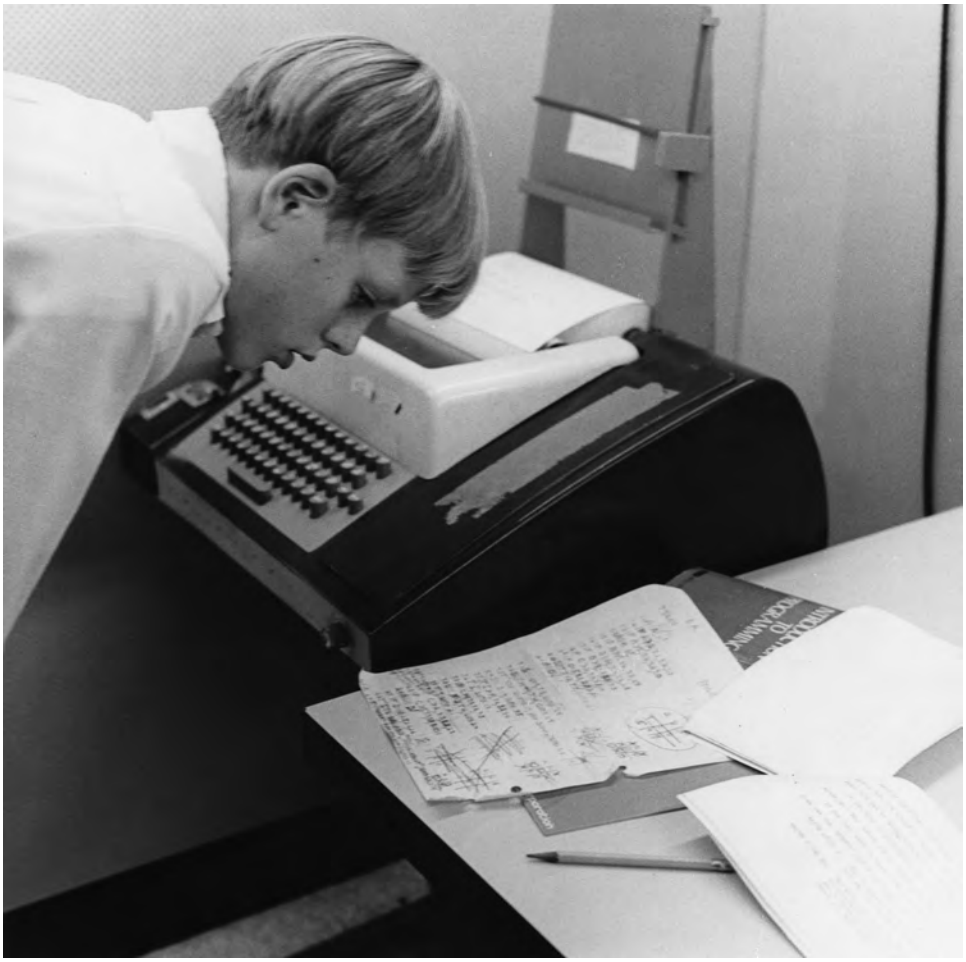


Figure 4.5 **A school-aged boy examines notes for a BASIC program near a teletype machine. Next to the code listing is a book entitled "Introduction to Programming."** (Courtesy of the Computer History Museum)

rejected most of the social and cultural aptitudes as little more than facts about computers—syllabi items memorized one moment and forgotten the next. What Luehrmann yearned for were *performance objectives* related to conceptual thinking and programming, and *cognitive skills* that students could activate and enhance by doing. The only learning objectives he accepted were those related to *problem solving and coding*, such as developing algorithms, designing data structures, detecting logic errors, and modifying programs to accomplish important tasks. His overall assessment of the report was blunt: "I will argue that fully four fifths of these

empirically discovered objectives should not be used in any significant definition of computer literacy. The teacher who is teaching toward that test instrument is teaching the wrong things."[31]

Anderson, Klassen, and Johnson offered a brusque written reply that was printed in the same December 1981 issue, immediately after Luehrmann's letter. The co-authors didn't back down, but insisted that there were two common definitions of the word *literacy* in the English language. Their approach to computer literacy embraced *both* meanings. Literacy can be defined as "the ability to read and write" (i.e., *to do*), but the word can also mean "the state of being informed, cultured, and well versed" (i.e., *to know* something).[32] This is exactly why the authors proposed a comprehensive view of computer literacy, and why many teachers in America were already at work introducing new subjects. Many believed that it was essential for students to learn about the computer's *role in society* as well as to perform useful work with the electronic devices. Making simple calculations, using popular applications, and programming had their uses. But there was more at stake. Anderson, Klassen, and Johnson proposed their own analogy: Just as *scientific literacy* was generally understood to mean both knowing scientific facts and understanding the broader implications of science-and-society issues, so *computer literacy* should include both practical skills and an appreciation for the role of computers in society.[33] The Minnesota scholars concluded with a rebuttal designed to push BASIC programming to the margins of the K-12 curriculum:

> Some of these things [about computer systems] can be learned as a byproduct of learning to write simple BASIC programs, but most of this type of useful knowledge cannot be learned that way. Indeed we would argue that most of what every ordinary citizen needs to know about computers will not be learned from learning how to program.[34]

Anderson, Klassen, and Johnson did acknowledge that the academic study of computer science was taking place in colleges and universities, but they hoped to keep well away from this ground in K-12 contexts. In their opinion, computer literacy and computer science were two separate realms. "The most succinct distinction is to say that computer literacy is that part of computer science that everyone should

31. Luehrmann, "Computer literacy — What should it be?" 682.

32. Ronald E. Anderson, Daniel L. Klassen, and David C. Johnson, "In defense of a comprehensive view of computer literacy – A reply to Luehrmann," *The Mathematics Teacher* 74, no. 9 (December 1981): 687–690, here at 687.

33. Anderson et al., "In defense of a comprehensive view," 687.

34. Anderson et al., "In defense of a comprehensive view," 688.

know or be able to do... Computer literacy should be thought of as the knowledge and skills the average citizen needs to know (or do) about computers."[35]

4.6 A Blow to the Movement

Both sides in the education debate had clearly drawn their daggers, sensing there was a lot at stake in the controversy over computer literacy. But it appears that the opinions of Anderson, Klassen, and Johnson had the most impact, dealing a blow to the learn-to-program movement in America's K-12 schools. Computer education would have to make room for other subjects, and administrators were reluctant to require coursework in Logo or BASIC across the board, particularly in contexts where it did not arise naturally. After all, purchasing enough computers to have a comprehensive impact on computer literacy would be tremendously expensive. By the late 1980s and early 1990s, the academic study of computer science in high schools would recede almost completely from view. In its place were basic literacy classes, word processing, and some exposure to databases and graphics programs.

For his part, Arthur Luehrmann was unrepentant, and he continued doing what he could do to teach programming skills to the American public. In the December 1981 issue of *Popular Computing* magazine, Luehrmann summarized what became his final position, that computer literacy must be equated with programming skills. "The goals of a computer literacy program are to teach programming and programming skills, new ways of thinking, planning skills, and debugging strategies."[36]

The whole debate seems to have launched Arthur Luehrmann on a new trajectory. A few years earlier, Luehrmann had moved from Dartmouth College to the Lawrence Hall of Science at the University of California, Berkeley. Now, to further his teaching agenda, Luehrmann co-founded Computer Literacy Press in Berkeley with Herbert D. Peckham and Martha Luehrmann. Peckham had established credentials as an author of BASIC programming books, and Martha Luehrmann (Arthur's wife) took on important administrative and leadership roles at the new company, eventually serving as its President. The Bay Area publishing house became a successful partnership that produced a line of useful programming primers for schools and self-taught programmers.

Apple Computer had an early lead in the educational marketplace, so Computer Literacy Press focused on the Apple II platform first, and they established local connections in the Bay Area. The team published *Apple PASCAL: A Hands-on*

35. Anderson et al., "In defense of a comprehensive view," 687–688.

36. Luehrmann quoted in Albert Benderson, "Computer Literacy," *Focus* 11 (Princeton, NJ: Educational Testing Service, 1983): 6.

Approach (1981), followed by *Computer Literacy: A Hands-on Approach* (1983), which introduced computers through BASIC programming on the Apple II.[37] Apple Computer also appreciated the group's efforts, and for a time they distributed a copy of Luehrmann's *Apple PASCAL* book with every unit of the Apple Pascal software that they sold.

Co-marketing arrangements like this would become a common feature of the software industry before the arrival of the commercial Internet. For a new software product to make inroads into a busy marketplace before then, it was good publicity (and helpful product support) for a company like Apple to sponsor books and articles that explained how their software worked and what the hidden or challenging features might be. If the books were written and produced by third-party authors, customers would be more likely to trust their opinions and feel that they were not simply reading advertisements for the company's products. Computer Literacy Press, the Waite Group, and other content providers were quick to recognize this, and they expanded their offerings by employing authors who had experience writing for technology companies.

4.7 Apple Computer's Education Agenda

Did the computer literacy movement influence the sales of PCs in schools? The short answer is "yes," but only gradually. According to the National Center for Education Statistics, at the beginning of the 1980–1981 academic year there were approximately 31,000 PCs and 22,000 time-sharing terminals available to elementary and secondary students in the U.S. By the beginning of the 1982–1983 academic year, the number had risen to 96,000 PCs and 24,000 terminals.[38] These statistics show a steady three-fold growth in units, but not the explosive growth seen in the home computer market. In fact, throughout the 1980s and 1990s, many American schools continued to have only a few computers per class available for student use. Often, the devices were chained to carts or locked in rooms, and students had limited access to them. The visions of deep learning and ubiquitous computing articulated by Ted Nelson, Seymour Papert, and Lee Felsenstein were still dreams for the future.

Apple Computer recognized the gap between home use and school use, and became one of the early PC firms to establish an educational outreach program

37. Luehrmann and Peckham, Apple PASCAL: A Hands-On Approach; Arthur Luehrmann and Herbert D. Peckham, *Computer Literacy: A Hands-On Approach* (New York: McGraw-Hill, 1983).

38. Albert Benderson, "Computer Literacy," *Focus* 11 (Princeton, NJ: Educational Testing Service, 1983): 3.

for their products. Through the initiative, they encouraged a broad-based computer curriculum, and they worked to accelerate the pace of Apple II adoptions in schools.[39] Apple was especially eager to capitalize on the growing demand for computers in the middle school and high school markets. Between 1980 and 1983 the Educational Marketing Director at Apple was Glenn Polin, a transplant to the West Coast who had earned a B.S. degree in Psychology and an M.S. degree in Computer Science from University of Illinois at Urbana-Champaign. Polin helped to launch the "Apple Seed" computer literacy program in 1981, which distributed kits containing software, reference materials, and workbooks with exercises for junior high and high school students. Schools received the kit for free if they had recently purchased a 32K Apple II computer, complete with disk drives.[40] The campaign was moderately successful, reaching 700 school systems across the U.S. and Canada.

Apple was also doing well in its campaign to build retail outlets for its products. In 1981, the company launched an "Apple Expo" initiative aimed primarily at attracting new Apple dealers and building customer networks in major hubs across the country. Like a modern-day circus, the Apple Expo consisted of a "big-top tent" filled with product showcases, multimedia presentations, and a diverse range of customer experiences. The buildings and trade show booths were transported around the country via a fleet of semi-trucks, which pulled into the key markets of Dallas, New York City, Chicago, and Los Angeles during March and April of 1981. Steve Jobs, Mike Markkula, and other Apple spokespeople gave presentations on hardware and software products. They were assisted by peripheral manufacturers and representatives from several software companies who sought to partner with Apple.

Admission to the Apple Expo was $10 per person at the door, but Apple supplied authorized dealers within a 100-mile radius of the event with free tickets for customers who wanted to visit the exhibits. *Softalk* magazine described the high-tech showcase as the modern-day equivalent of the Ringling Brothers Barnum and Bailey Circus coming to town, "although the magic it presents is far more amazing than circus feats." Instead of three rings, however, the main Apple building housed "six wedge-shaped theaters, like slices of an apple pie." According to the report, the

39. In the mainframe and minicomputer worlds, computer manufacturers had attempted this work for some time. For example, the Digital Equipment Corporation (DEC) launched an EduSystem program in the early 1970s that distributed books and software to schools. Some DEC employees even delivered the kits using a custom EduVan. See John J. Anderson, "Dave tells Ahl: The history of creative computing," *Creative Computing* 10, no. 11 (November 1984): 66–74, here at 70.

40. "Apple launches literacy program," *InfoWorld*, March 30, 1981, 11.

star was neither trapeze artist, wild animal trainer, nor clown: "it's a computer," the group enthused.[41]

4.8 Applications over Languages

In the early 1980s, Apple's sales and marketing push was about hardware platforms and applications, not programming languages. Like the computer literacy debates, this trend worked against the learn-to-program movement in its educational manifestation, because it took away the urgency to learn to code as a strategy to create useful software. Glenn Polin voiced these general concerns when he argued *against* the general introduction of programming skills in a national report about the K-12 curriculum. Polin emphasized the value of *applications over languages* for most students and office workers. He wrote:

> Some educators place an undue emphasis upon teaching students to program computers. We are all users of devices, such as the television and telephone, and a user of a device doesn't need to understand everything about how the technical aspects of the device work. Do we want everyone to be programmers? I think the answer to that in the future is going to be increasingly no... Increasingly, future applications will be user-friendly and will not require an intimate knowledge of the computer.
>
> The software evolution of the last three or four years bears that out. We have packages on the market today that can be used by a total novice, by a secretary with no training, by a manager with no training, and that trend is accelerating. When you go out into the world, you're not going to be asked to do basic programming unless you choose that as a vocation.[42]

Polin's comments run parallel to Apple's "big tent" approach to selling hardware and software, and they seem to be directed *against* programming advocates like Arthur Luehrmann, Bob Albrecht, and Seymour Papert, who viewed coding activities as valuable cognitive skills, whether or not a person chooses to become a professional programmer. Polin and his team did successfully deploy the Apple II in many schools, but the accompanying curriculum placed little emphasis on learning to program. Glenn Polin left Apple Computer in August 1983, 4 months before the release of the Macintosh.

Were computer scientists more sympathetic to the idea that computer literacy should include at least *some* exposure to BASIC or a similar language? The short answer is no. ACM Fellow and Turing Award Winner Butler W. Lampson of the Xerox

41. "Marketalk news," *Softalk* 1 (April 1981): 59.

42. Quoted in Benderson, "Computer Literacy," *Focus* 11: 6.

Palo Alto Research Center probably knew as much as anyone in his era about the task of software construction. However, in a 1986 interview he suggested that learning to program in BASIC was a waste of time, and that it was irrational to think the language would survive into the 21st century. I quote here from his important conversation with Susan Lammers about software development in the book *Programmers at Work*:

> Lampson: "I think the idea of computer literacy is such a rotten one. By computer literacy I mean learning to use the current generation of BASIC and word-processing programs. That has nothing to do with reality. It's true that a lot of jobs now require BASIC programming, but the notion that BASIC is going to be fundamental to your ability to function in the information-processing society of the twenty-first century is complete balderdash. There probably won't be any BASIC in the twenty-first century."
> Interviewer: "So how should we prepare ourselves for the future?"
> Lampson: "To hell with computer literacy. It's absolutely ridiculous. Study mathematics. Learn to think. Read. Write. These things are of more enduring value. Learn how to prove theorems: A lot of evidence has accumulated over the centuries that suggests this skill is transferable to many other things. To study only BASIC programming is absurd."[43]

This excerpt on literacy and programming is only part of a much larger conversation about future trends in computing and the role of PCs in society. Lampson was responding to the anxiety that many parents felt when they worried that their children "won't have a future if they don't learn to program in BASIC."[44] This unease seems to be a reflection of how the computer literacy debates were percolating across American society in the 1980s. Very correctly, parents were wondering what the impact would be on their children if they did *not* acquire the necessary skills for success in the dawning computer age. In recent times, we have also recognized the importance of equal access to computing fundamentals, especially opportunities for girls and underrepresented minorities, who were left behind by many coding initiatives. From the mid-1980s onward, disparities in educational opportunities have furthered the message that Computer Science is a field that belongs to a limited range of students.[45]

43. Susan M. Lammers, ed., *Programmers at Work* (Bellevue, WA: Microsoft Press, 1986), 37–38.

44. Lammers, *Programmers at Work*, 38.

45. Jane Margolis, Rachel Estrella, Joanna Goode, Jennifer Jellison Holme, and Kimberly Nao, *Stuck in the Shallow End: Education, Race, and Computing*, Updated Edition (Cambridge, MA: The MIT Press, 2017), 51.

Ambivalent attitudes about teaching programming in schools is only part of the story, however. Outside of academic circles, programming gained momentum as a profitable and pleasurable activity, sometimes in the most unlikely places. Chapter 5 explores several of these new directions, including how updated versions of the BASIC language surged in popularity among computer gamers, self-taught hobbyists, and information technology (IT) professionals. This new phase of the learn-to-program movement gathered its energy not from schools or educational agendas, but from the commercial activities of corporations like DEC, Microsoft, and Borland International. Although the learn-to-program movement would lose its intellectual footing as a campaign centered on education and cognitive development, it would continue in popular and commercial manifestations, culminating in a user base of millions of developers.

5 Four Million BASIC Programmers

"Most of the games in this book require no special knowledge, tools or equipment to play, except, of course, a BASIC-speaking computer."

David Ahl, *101 BASIC Computer Games* (1973)

"Microsoft BASIC had hundreds of thousands of users around the world. How are you going to argue with that?"

Don Estridge, team leader for the IBM Personal Computer[1]

The deliberations about programming and computer literacy would swirl for years, but in some sense, they didn't matter: the first personal computers (PCs) brought tens of thousands of computers into homes and offices, and this critical mass helped software achieve new levels of distinction. As new interpreters and compilers arrived and were put through their paces, programmers pushed the systems to their limits. This chapter explores how the BASIC programming language gained new supporters from *outside* the academic world in the 1970s and 1980s, culminating in a community of some four million BASIC programmers. This surge also laid the groundwork for the introduction of a new tool, Microsoft Visual Basic (1991), which brought rapid application development (RAD) to the MS-DOS and Microsoft Windows platforms.

This chapter shows how programming skills were diffused in the U.S. not only by teachers and school administrators, but through the efforts of highly-motivated authors, publishers, and entrepreneurs. These men and women introduced programming skills to Americans from all walks of life, including computer gamers, hobbyists, students, office workers, power users, and information technology (IT) professionals. After self-taught programmers learned the ropes, they

1. Don Estridge quoted in Lawrence J. Curran and Richard S. Shuford, "IBM's Estridge," *Byte* (November 1983): 88–97, here at 88.

invested in new systems, built application programs and utilities, and gradually expanded the reaches of programming culture. BASIC deserves special mention in this important manifestation of the learn-to-program movement. As *PC Magazine* editor Bill Machrone wrote in 1983, "The real hero of the personal computer revolution is BASIC. Microsoft's prodigal program has coaxed out the programmer hidden within so many of us that it is and will continue to be an integral part of the personal computer scene for many years to come."[2]

This chapter also introduces the book publisher Microsoft Press, a division of Microsoft established in 1983 to produce high-quality computer books for the leading PC platforms. Microsoft Press joined several other computer book publishers in producing hundreds of books each year about the principal hardware and software systems of the PC era. I focus on primers about BASIC and Visual Basic in this chapter, but you'll see dozens of titles from Microsoft Press and other publishers in the chapters that remain. My argument is that computer books should be seen as important bellwethers of the learn-to-program movement. These resources filled the gaps in America's technical education system, especially for self-taught programmers who learned the ropes on their own and gradually entered professional computing occupations. Computer books carried with them the values and commitments of the era's software creators, consumers, and entrepreneurs. They conveyed the industry's best practices and prominent computing myths, including the aspirations and ambiguities about programming that had existed in the industry since the 1960s.

5.1 Introducing David Ahl

The rising tide of popular support for BASIC can be measured by the success of a pivotal entrepreneur and book author, David H. Ahl (1939–), likely the first technical writer to sell over a million copies of a programming primer in the U.S. (See Figure 5.1.) Ahl started publishing computer books about mainframe and time-sharing BASIC in 1973, and he continued to write bestsellers about BASIC in the era of personal computing. His titles were eagerly consumed by students, teachers, hobbyists, and industry professionals alike. Like Daniel McCracken and Bob Albrecht, Ahl paved the way for a new style of technical writing, the personality-driven primer, which emphasized the author's style and voice, and sought to create a lasting bond with the reader. Ahl's project's also kick-started one of the most lucrative segments of the software world in the 1970s and 1980s, *computer games*. The success of this genre provides evidence that Americans were building a programming

2. Bill Machrone, "Linguistics: Languages for the PC," *PC Magazine*, September 1983, 115–145, here at 141.

Figure 5.1 **David Ahl in the *Creative Computing* booth at the West Coast Computer Faire, Civic Center, San Francisco (1980).** (Courtesy Jim Warren and the Computer History Museum)

culture around entertainment and new technologies in the 1970s and 1980s. This enthusiasm for gaming would soon overflow into new computing platforms and products.

David Ahl began his career at the Digital Equipment Corporation (DEC) in 1970. The innovative computer company was housed in a former woolen mill in Maynard, Massachusetts, and Ahl spent his early years working as an education manager, marketing DEC computers to schools. In 1971, while working with high school and college students around the country, Ahl developed the idea of collecting text-based computer games from the people he met and distributing them. Ahl had been deeply impressed with how creative young BASIC programmers were, and he thought that sharing games would be a fun way to build the DEC platform and establish a user community. Ahl put out a call in DEC's *Edu* newsletter for students and faculty to mail their favorite BASIC games to him, and he promised to share the results.

Ahl was astonished with the feedback that he received from BASIC users at home and abroad. Hundreds of letters poured in with source-code listings and enthusiastic notes from game players of all ages. Some of the programs arrived from

middle school and high school students, writing or adapting their first programs. Others came from prominent educators and coding advocates like John Kemeny, Bob Albrecht, and Arthur Luehrmann. Numerous regions in the country were represented, not just the computing hotbeds of New England and Silicon Valley. Contributions arrived from 17 states, including California, Connecticut, Delaware, Georgia, Illinois, Massachusetts, Minnesota, New Jersey, New York, Oklahoma, Ohio, Oregon, Pennsylvania, Rhode Island, Tennessee, Texas, and Virginia. International correspondence even arrived from programmers in Canada, the U.K., and the Netherlands.

David Ahl carefully curated the games on his own time, testing the programs, rewriting them if necessary, and verifying that they ran on DEC time-sharing systems. For a software platform, Ahl had in mind versions of DEC BASIC that were distributed via the EduSystem marketing program in the early 1970s. He also shared some of the games through the Digital Equipment Computer Users' Society (DECUS), an organization dedicated to sharing software among DEC enthusiasts.

When everything was ready, Ahl assembled a manuscript. He selected the most interesting games, put them in alphabetical order, and printed a volume that he titled *101 BASIC Computer Games*, which DEC distributed in 1973.[3] Each program featured a description, notes about the game's usefulness, a list of system limitations, and the name and address of the contributor, when known. (Some of the games had unknown origins or multiple sources.) He also acknowledged when the games started out as FORTRAN or LISP programs, and were later converted to BASIC. Several prominent computer centers appear in his notes, including Dartmouth's Kiewit Computation Center (Hanover, NH), the Lawrence Hall of Science (Berkeley, CA), People's Computer Company (Menlo Park, CA), Lexington High School (Lexington, MA), and Oregon Museum of Science and Industry (Portland, OR).

In his Preface, Ahl claimed that *101 BASIC Computer Games* presented the first collection of games written and published entirely in BASIC. He also claimed that the games offered substantial instructive benefits: "the educational value of games can be enormous—not only in their playing but in their creation."[4] The notes indicated that many of the games began as high school or college projects, and then the instructors used them for further explorations in word puzzles, poetry, algebra, random numbers, dice, board games, card games, the laws of motion, and graphic design. (See Figure 5.2.)

3. David H. Ahl, ed. *101 BASIC Computer Games* (Maynard, MA: Digital Equipment Corporation, 1973).

4. Ahl, *101 BASIC Computer Games*, 7.

Figure 5.2 **Students collaborate on a coding project using a teletype machine to enter and test a program.** (Courtesy of the Computer History Museum)

A few samples from the collection will survey the range of programs that early BASIC programmers were creating. John Kemeny from Dartmouth sent in a game called Battle of Numbers (a variation of the game Nim), where the user and the computer alternately removed objects from a virtual pile. The player taking the last object from the pile is the winner—a determination made by the program's calculations in modulo arithmetic. The source code was 90 lines long and used the BASIC keywords PRINT, FOR… NEXT, INPUT, IF, GOTO, LET, GOSUB, RETURN, and END.[5]

Lexington High School student Jim Storer (Lexington, MA) sent in a version of the popular game Rocket, which allowed the programmer to simulate landing an Apollo capsule on the moon, a celestial occurrence that greatly fascinated the American public. The simulated astronaut started 500 feet above the lunar surface and controlled the burn rate for his or her landing rockets as they attempted to land. The computer rockets could fire in one-second bursts, and each unit of fuel

5. Kemeny in Ahl, *101 BASIC Computer Games*, 32–33. Kemeny did not use the Mod operator, because it not available in early versions of BASIC. Instead, he used the formula $P = Q - C * INT (Q / C)$.

slowed the decent of the capsule by one foot per second. Jim Storer added drama to his program by using PRINT statements and clever mathematical calculations that simulated the forces of gravity. To heighten the drama, the user was told that the on-board computer for the Apollo landing capsule had temporarily failed, so the gamer needed to land the craft manually. During free fall, the goal was to continually adjust the burn rate so that the craft slowed gradually without running out of fuel. If the mission failed, the message "You Blew it!" was displayed, along with data about the impact velocity of the capsule and the depth of the resulting moon crater.[6]

Computer gaming was tremendously interesting for young programmers, as it is today, but in the early 1970s gaming was not yet a commercially viable market. In fact, when Ahl approached DEC management with the idea of publishing his game collection in 1973, the company resisted the project. Ahl was insistent, however, and DEC eventually agreed to publish the first edition of the book while suggesting the author take later editions elsewhere. The price of *101 BASIC Computer Games* was set at $7.50.

As part of his agreement with DEC, Ahl was given the right to keep royalties for future editions, if any were published. Retaining these royalty rights was a shrewd move for Ahl, who believed in his concept and hoped to revise and reprint the book several times. This bolstered Ahl's credibility as a writer and entrepreneur, and the revenue he earned supported future publishing projects, including *Creative Computing* magazine, which he founded in 1974.

In 1978, the second edition of Ahl's book came out, entitled *Basic Computer Games: Microcomputer Edition*. (See Figure 5.3.) This book included enhanced games revised for Microsoft BASIC on several early PC platforms.[7] In 1979, Ahl published a third book, *More BASIC Computer Games*, a new compilation containing 84 additional program listings that had been collected from contributors across the country. Both new books benefited from the coming of PCs, which added thousands of hobbyists and early adopters to the marketplace. In later interviews, Ahl claimed that the *Computer Games* series sold over one million copies—an astonishing number with historical significance.[8] The scope of this user base indicated that programming was no longer the domain of just academics and engineers, but it

6. Storer in Ahl, *101 BASIC Computer Games*, 182–187. Due to the game's popularity, two additional versions were included in the book—one by William Labaree of Alexandria, Virginia, and a second by Eric Peters, a DEC employee from Maynard, MA.

7. David H. Ahl, ed., *BASIC Computer Games, Microcomputer Edition* (Morristown, NJ: Creative Computing Press, 1978).

8. For information about the book's sales and the founding of *Creative Computing* magazine, see John J. Anderson, "Dave tells Ahl: The history of creative computing," *Creative Computing* 10, no. 11 (November 1984): 72, 81. For an image of the magazine's cover, see Figure 11.2.

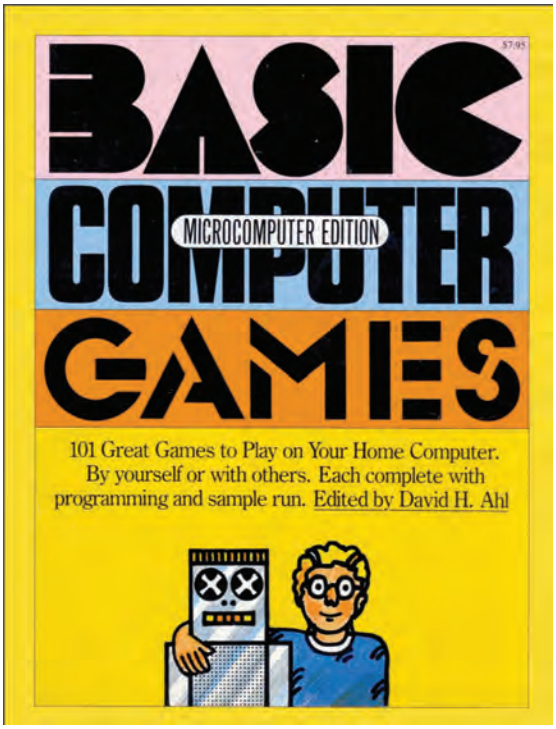


Figure 5.3 **Cover of David Ahl's *BASIC Computer Games, Microcomputer Edition*** (Copyright ©1978. Used by permission of Workman Publishing Co., Inc., New York. All Rights Reserved)

had become a popular pastime in America. Despite the criticism that BASIC had received from some authorities in the computing world, it was the language that most users knew.

David Ahl's books also highlight an important dynamic of information flow in the years before the commercial Internet. To learn about their systems, programmers in the 1970s and 1980s bought computer books and magazines. The authors of these publications became vital agents in the diffusion of technical information, because they united software developers in virtual communities and provided them with needed instruction when they had few other resources. Company manuals and user group newsletters played important supporting roles, too, as did (on occasion) the mainstream media of television, film, and radio. But printed books and magazines should give historians special pause, because they communicated shared ideals and presented skills that were deemed necessary for entering one or more computing subcultures. These groups included gamers, power users, systems programmers, and professional developers who strongly identified with an operating system like CP/M, Unix, MS-DOS, Windows, or the Macintosh.

5.2 A Proliferation of BASICs

Let's quickly recap the twists and turns of the BASIC language before and after the development of PCs.

First, BASIC was created as a new high-level language by scientists at Dartmouth College in the mid-1960s, and it became a teaching language closely associated with math and science instruction, expanding into a niche originally occupied by FORTRAN.

During the computer literacy movement of the 1970s, thousands of students experimented with BASIC in high school and college classrooms to make their first steps with computers. Computer literacy advocates like Bob Albrecht and Arthur Luehrmann equated BASIC programming with cognitive development and learning to control and use computers. These instructors began their careers with time-sharing BASIC hosted on mainframes and minicomputers, and gradually switched to PCs when they became cost-effective for students and self-taught programmers. As the first microcomputers arrived in the mid-1970s, BASIC was selected as the common language of most platforms. This was became small BASICs could be squeezed into the tiny memory resources available, but also because BASIC reflected the populist ethos of personal computing for many. In counterculture America, there were some who argued that FORTRAN was too closely connected to corporate and military computing contexts.

The first PC BASICs are now referred to now as "classic BASIC" by historians. Classic BASICs are distinguished by their small size in memory, implementation as an interpreter, rudimentary flow control mechanisms, use of line numbers in code, and a lack of structured programming elements. The next wave of the BASIC language is sometimes described as "structured BASIC" or "second-generation" BASIC. These products were sold in compiler and interpreter versions, and, beginning in the mid-1980s, they were available for both time-sharing systems and PCs. Structured BASIC typically offers enhanced graphics capabilities, improved language features, support for user-defined procedures and flow control elements, and an integrated development environment (IDE) for program construction and debugging.

Not all BASICs fit easy categorization, however. An interesting implementation that deserves more attention is HP BASIC, a family of products created by Hewlett-Packard (HP) for their computers and calculators. One of the earliest versions, HP Time-Shared BASIC, received wide distribution in the late 1960s and 1970s, and it did a lot to popularize BASIC programming in schools. California high school student Mike Mayfield used this implementation to create his popular Star Trek computer game in 1971, and the game received its start on an HP 2000C time-sharing

computer. Star Trek and Super Star Trek became two of the most popular computer games in the 1970s, and they have enjoyed a resurgence in the "retro" game market in recent years. David Ahl helped to popularize these HP BASIC versions in several of his programming books.

In the early 1980s, HP BASIC was further adapted for a new line of HP engineering workstations. These computers shared some of the attributes of early IBM PCs and compatibles, but they were higher end and more expensive. For example, the HP 9836A computer system debuted in 1981, priced at $11,950. This system included a flat-top monitor, two 5.25" floppy disk drives, eight I/O slots, and a Motorola MC 68000 processor. A new version of BASIC was designed for the system, and it contained structured programming elements, advanced graphics capabilities, function keys, and a menu-driven IDE.

The engineer who led the BASIC development team at HP was Kathryn Kwinn, a Ph.D. in Computer Science from Iowa State University. Kwinn joined HP in 1978 and became the first woman from the HP desktop computing divisions to author or co-author a technical article in *Hewlett-Packard Journal*. Her 1982 technical brief describes the newest features of HP BASIC, HPL, and Pascal, and how these languages were optimized for the HP 9826A/9836A systems.[9] These were important desktop computers for their era, but they suffered from a high retail price point. Nevertheless, the HP BASIC systems were well designed and they point to the diversity of computing contexts in the evolving hardware and software industries.

5.3 IBM BASICA

In early 1981, Microsoft BASIC was still the best-known version of the language available for use on microcomputers. However, the situation became more complicated when IBM released the IBM Personal Computer in August of 1981. Although the early IBM PCs contained a version of Microsoft BASIC called BASICA (for Advanced BASIC), many of the later PC "clone" manufacturers chose to develop their own versions of the interpreter. IBM could have followed this second strategy, because they had a very successful version of BASIC for mainframe computers that they might have ported to their new device. However, the company took a cautious "off the shelf" approach when sourcing most of the IBM Personal Computer's first components, including its software. (See Figure 5.4.) Don Estridge, team leader for the first IBM PC, summarized the selection of Microsoft BASIC in this way: "Microsoft BASIC had hundreds of thousands of users around the world. How are

9. See Kathryn Y. Kwinn, Robert M. Hallissy, and Roger E. Ison, "The 9826A/9836A language systems," *Hewlett-Packard Journal* 33, no. 5 (May 1982): 24–32.

Figure 5.4 **The IBM PC-DOS 2.02 product documentation (1983), complete with a BASIC user's guide written by Microsoft.** (Photo: Herb Bethoney. Courtesy of the Computer History Museum)

you going to argue with that?"[10] So IBM engineers put BASICA in computer ROM and reserved the BASICA acronym for later use.[11]

Many IBM PC-compatible manufacturers also chose to license Microsoft BASIC. When they did so, Microsoft gave each original equipment manufacturer (OEM) a program known as GW-BASIC for their systems. GW-BASIC debuted in 1983 and the interpreter was distributed as part of the MS-DOS operating system. GW-BASIC was essentially identical to IBM's BASICA, with a few technical differences related to how the program was installed. GW-BASIC came with a character-based IDE like the

10. For the quote and an insightful interview with Estridge, see Lawrence J. Curran and Richard S. Shuford, "IBM's Estridge," *Byte* (November 1983): 88–97, here at 88.

11. For more about IBM's strategy when developing the first IBM Personal Computer, see James W. Cortada, *IBM: The Rise and Fall and Reinvention of a Global Icon* (Cambridge, MA: The MIT Press, 2019), especially chapter 14, "'A Tool for Modern Times:' IBM and the Personal Computer."

ones in use at HP and Dartmouth College. This environment allowed the programmer to use direction keys to move the cursor around the screen to edit the program currently in memory. The IDE also supported function keys and keyboard shortcuts, which software developers could use to edit and test their programs. It was a big improvement over the Altair implementation of BASIC and the first time-sharing systems.

New microcomputer BASICs soon arrived from Amiga, Apple, Atari, Casio, Commodore, Compaq, Radio Shack, Sinclair, Texas Instruments, and other manufacturers. Many of these products were closely related to Microsoft BASIC, and others created their own implementations that exploited their unique hardware, graphics capabilities, or operating systems. Charting all these BASICs is beyond the scope of this book, but it stands as a *desideratum* for future historians who are working to assess the diffusion of low-cost programming systems as the software industry took shape. What do all these BASICs have in common, and what makes each version unique? How did organizations and customers in the global marketplace respond to them?

One emerging trend seemed obvious. By 1983, it seems that many American consumers wanted to use BASIC to write their own text-based computer games. In doing so, they built on a venerable tradition that began in the 1970s: *adventure games*, an interactive computer quest driven by puzzle solving.

5.4 Adventure Games

"You are standing at the end of a road..."[12]

Part of the excitement of visiting early computer centers in America were the small crowds that would gather in the evenings as students or co-workers played a computer game. This social activity often took place at night, when most of the administrators and buttoned-down types had gone home. Since the debut of *Spacewar!* in the early 1960s, it has been common to see systems programmers, students, hackers, and spectators huddled around a computer screen enjoying gaming activities. In the mid-1970s, the computer gaming pastime expanded with the addition of a new genre, "adventure games," which placed a computer user in an interactive story driven by puzzles, hand-to-hand combat, and the exploration of mysterious spaces. The first adventure games were text-based simulations in which a user moved through a series of complex virtual challenges, often culminating in battles with mythological creatures. Innovative computer games in the genre included

12. Quotation from the opening screen of Will Crowther and Don Woods original *Colossal Cave Adventure* (1976).

Figure 5.5 **Zork II software package for the Apple II computer (1981–1983), sold by Infocom, Inc. Zork and other "adventure" games were a big draw on the first PC platforms.** (Courtesy of the Computer History Museum)

Hunt the Wumpus, developed by Gregory Yob in 1973; *Colossal Cave Adventure*, created by Will Crowther and Don Woods in 1975–1976; and *Zork*, created by Tim Anderson, Marc Blank, Bruce Daniels, and Dave Lebling in 1977. (See Figure 5.5.) When PCs arrived, adventure games were also one of the biggest draws to the new platforms.

The original adventure games were programmed on commercial minicomputers, and they were designed to be played in the time-sharing environments typical of research labs and university computing centers. Will Crowther created his game *Colossal Cave Adventure* using FORTRAN on a DEC PDP-10 minicomputer.

This virtual world was reminiscent of *Dungeons & Dragons*, with a physical layout taken from a cave in Kentucky where Crowther had spent time spelunking. Typical game play in this type of digital landscape involved issuing directional commands on a keyboard indicating where an adventurer should go. Gamers could also type English words to indicate what their character might pick up, where they should move to, or how they should solve puzzles. Within the program code, the adventure game logic involved a heavy dose of text processing, i.e., using *parsers* to translate the adventurer's textual input into action commands for the game. Adventurers typically needed to open locked doors, find essential items, and build mental maps of abstract spaces (like caves or castles). In the most elaborate games, the goal was to complete a pre-defined *quest*, which might have distant literary connections to medieval legends or even crusade literature. In some adventure games, like *Zork*, it was also possible to score points in connection with game progress. If you were skilled, you could attempt to set the high score on the system.

As the personal computing era took shape, many PC programmers tried their hand at writing or editing adventure games, often using BASIC because the language was readily available and easy to use. One of the most important promoters of adventure gaming on the IBM PC/MS-DOS platform was *PC Magazine* columnist and science fiction writer Dian Crayne (1942–2017), a skilled software developer who also taught hobbyists how to build their own adventure games. Crayne published her adventure titles with various software publishers, gaining experience and credibility with each new software release. She was an expert in the text-based, story-driven gaming genre, which was a good fit for her technical and literary abilities.

Crayne was an early employee at Xerox and Norell Data Systems. At Norell, Crayne worked as a programmer and game designer, contributing to *The Phantom's Revenge*, *The Hermit's Secret*, *Monster Rally*, *Valley of the Kings*, and *Elsinore*. These were text-based adventure games with an extensive story line but little in the way of computer graphics. As such, they were a good fit for the bare-bones PC systems that were popular in the late 1970s and early 1980s. Each gaming title required literary creativity, as well as technical precision.

In 1983, Crayne published a feature article in *PC Magazine* explaining how she designed and created her adventure games, inviting readers to take part in the fascinating worlds of gaming strategy, resource allocation, text parsing, and plot twists. Crayne was an accomplished assembly language programmer, and she coded in assembler for most of her commercial games. This allowed her to use memory resources efficiently, and it also created a fast and responsive gaming experience for users of the original IBM PC. But Crayne also recognized that PCs were gaining power and capacity, and this could open up game development for the hobbyist

who had a moderate level of proficiency in BASIC. The biggest issue was space for all the data that was needed. Crayne advised:

> Most of the larger text adventures are written in assembly language for the sake of speed and space conservation. A "large" game is one that has over 200 room locations and over 50 objects. If a game isn't too large, and if speed isn't too much of a consideration, it can be written in any other language—including BASIC.[13]

Crayne then went on to describe how a programmer could organize their adventure, introducing the significance of vocabulary words and a parser:

> My own adventure games are built from two basic parts: the driver program and the text files or "script." The script contains all of the vocabulary words that the driver recognizes, plus the object and place descriptions. There is also a builder program that converts the text in the script to machine-readable tables. Because the games are script-driven, I can build 70 to 80 percent of a new game without ever touching the actual program source code…
> When you set out to write a parser, you enter the whole complex world of artificial intelligence. Fortunately, adventure games take place in a confined universe, such as a set of caves, where the author can control the objects and the actions… Because of this limitation, it is possible to read each word that the player types in, locate it in a table, and decide what to do with it.[14]

Vocabulary lists indicated which words were to be accepted in the game as directives, and Crayne recommended subdividing the vocabulary list into motion verbs, action verbs, nouns, function words, adjectives, and other semantic categories. In BASIC, Crayne recommended that programmers configure arrays for each of the vocabulary list categories, using the DIM statement to declare the arrays and DATA statements to fill the arrays with information. Subroutines could then be constructed to receive and validate the input from the user, and store game state information in one or more variables. Crayne reassured readers that only two types of input were really needed in an adventure game, movement requests and action requests. Input from the user could be tested easily and only a short list of actions would be required for the programmer to manage. Crayne concluded her article with a suggestion for BASIC source code that might be used to manage a game object like a cannon, which could be loaded and fired in a typical adventure program. As

13. Dian Crayne, "Do-it-yourself adventure," *PC Magazine*, September 1983, 266–276, here at 268.

14. Crayne, "Do-it-yourself adventure," 272.

noted above, Crayne was teaching users how to code *text-based* games, so the program logic was not related to computer graphics but textual output. Her routines instructed programmers how to display messages like, "KA-WHAM!" or "CANNON BALL hits the DOOR and smashes it to bits."[15] Crayne ended the lengthy article with a practical tip: it is best to start simple when writing computer games, to avoid frustration. She also encouraged BASIC programmers to add variety and complexity as they went—"everything from the Bible to nuclear physics."[16]

Dian Crayne was a skilled writer and software developer who moved fluidly from one subfield of computing to the next. After working at Xerox, Norell Data Systems, and *PC Magazine*, she turned to writing computer books for the users of new systems. In this genre, her published works included *The Essential User's Guide to the IBM PC, XT, and PCjr* (1984) and *Serious Assembler* (1985), the later which she coauthored with her husband, Charles A. Crayne. Dian was also associated with the science fiction writers Larry Niven and Jerry Pournelle, who collaborated with her on several fiction projects. (For more on Jerry Pournelle and his influences, see "Learn BASIC Now" below and information about his magazine columns in Chapter 8.) Crayne used several pen names to write her science fiction novels, including Dian Girard and JD Crayne. Her best-known books include the *Captain Spycer* series, published between 2005 and 2009, and *Murder at the Worldcon*, published in 2005.

Despite the success of computer gaming on the first PC platforms, IBM largely overlooked the gaming market on its first systems, envisioning their computers as business and educational tools, not sources of entertainment. Figure 5.6 provides visual evidence of their approach with the BASICA programming language and the IBM PCjr computer, released in 1984. In the publicity photo, a well-dressed male teacher lectures on BASIC to a small group of students who are seated in a classroom or computer lab. A program that simulates coin tosses is on the blackboard, but otherwise the mode of instruction is passive and focused on the authority of the instructor. The creativity of author-programmers like Dian Crayne is that they recognized people would learn best if they could create their *own* games and adventures, far from the controlling gaze of teachers and instructors. Her approach is much closer to the soul of the DIY learn-to-program movement, as it left classrooms and became associated with self-directed learning and hobbyist contexts.

5.5 Structured Programming

Devising complex adventure games did clarify one thing about classic BASIC—the rudimentary coding structures of BASICA or GW-BASIC did not make it easy to write

15. Crayne, "Do-it-yourself adventure," 276.

16. Crayne, "Do-it-yourself adventure," 276.

Figure 5.6 **Designed for the home and educational market, the PCjr featured a "Cartridge BASIC" version of the BASIC interpreter that could be inserted as firmware into one of the two slots on the front of the desktop computer. This staged image from 1984 depicts children learning BASIC at school. A short BASIC program that simulates coin tosses appears on the chalkboard.** (Courtesy of the Computer History Museum)

longer adventures. Moreover, early microcomputer BASICs lacked a full complement of the features that computer scientists *knew* a robust language should have, especially when longer, team-oriented projects were undertaken. As Paul Somerson asserted in an article about the state of BASIC programming in late 1983: "These days, 'It's written in BASIC' is a statement that is whispered, not shouted from rooftops."[17] For BASIC to continue as a conduit of the learn-to-program movement, it needed new features, including support for both *structured* design and *procedural* programming. These were interrelated coding paradigms that, if implemented successfully, might bring the language and its adherents into the modern age.

17. Paul Somerson, "In defense of BASIC," *PC Magazine*, September 1983, 328.

John Kemeny and Thomas Kurtz drew pointed attention to these missing features in their infamous book, *Back to BASIC* (1985), which attacked what they saw as the "corruption" of BASIC in the early 1980s and the language's urgent need for reform. The professors confessed that their own early versions of BASIC now seemed dated and rudimentary, but they claimed to have revised the language at Dartmouth, where they were putting the final touches on ANSI BASIC and a new commercial version entitled True BASIC. Both were "structured" forms of the language, they claimed, and vastly superior to the "street BASICs" that were now ubiquitous on low-cost PCs. The co-authors wrote:

> We are greatly concerned that a generation of students is growing up learning Street BASIC, an illiterate dialect of a lovely language. We feel that this is directly relevant to the problem that whereas schools now have hardware, educational software lags far behind. There have been devastating criticisms of BASIC in the literature. Unfortunately, as it applies to Street BASIC, we agree with them.[18]

Comparing Dartmouth College's "lovely language" to an "illiterate dialect" from America's streets smacks a bit like elitism. But in fact, the scholarly critique of unstructured languages was widely appreciated in academic circles, reaching a high-point in 1968 with Edsger Dijkstra's infamous letter to the Association of Computing Machinery (ACM), "Go To statement considered harmful."[19] Dijkstra argued that the use of unrestricted Go To statements in code complicated the task of analyzing and verifying the correctness of programs. In the worst case, Dijkstra argued, branching instructions like GoTo produced "spaghetti code," a tangled, unreadable mess that resembles the twists and turns of spaghetti noodles in a bowl. Although BASIC did not *intrinsically* have this problem, observers noted, it seemed that many students and hobbyists had rather unwittingly fallen into this way of implementing their algorithms in BASIC. For example, David Ahl's *101 BASIC Computer Games* made conspicuous use of GOTO statements, relishing in the simplicity of transferring program execution to any location in a source file when it was deemed necessary. Educational specialists who had studied programming and computer literacy were also beginning to document this problem through comparative studies of novice programmers.[20] The issue took on special urgency as the

18. John G. Kemeny and Thomas E. Kurtz, *Back to BASIC: The History, Corruption, and Future of the Language* (Reading, MA: Addison-Wesley, 1985), 56.

19. Edsger Dijkstra, "Go To statement considered harmful," *Communications of the ACM* 11, no. 3 (March 1968): 147–148.

20. For interesting examples related to BASIC, see Elliot Soloway and James C. Spohrer, eds., *Studying the Novice Programmer* (Hillsdale, NJ: Lawrence Erlbaum Associates, 1989); and Aqeel M.

learn-to-program movement gathered momentum as a grass-roots phenomenon, with thousands of young people learning coding habits from teachers, mentors, and authors who employed these techniques.

An example of "spaghetti code" in BASIC can be demonstrated with the following program, which uses classic BASIC syntax to display all the prime numbers under 100 on the screen:[21]

```
10 REM -- PRIME NUMBERS LESS THAN 100
20 N=N+1
30 IF N=100 THEN GOTO 120
40 I=1
50 I=I+1
60 J=N/I
70 IF INT(J)=J THEN GOTO 20
80 IF I>=SQR(N) THEN GOTO 100
90 GOTO 50
100 PRINT N,
110 GOTO 20
120 END
```

This valid "classic BASIC" program runs well and displays the prime numbers as directed, but it is certainly confusing to follow. The example comes from John Clark Craig's book on Microsoft Visual Basic programming, published in 1993 and discussed later in this chapter. The author used the routine to point out how easy it was to get lost using older ways of writing BASIC code. Fortunately for users, Craig maintained, Visual Basic had now changed all of this. "Learning how to create applications for Windows is now a breeze," he assured readers.[22]

By the mid-1980s, the major PC software companies had received the message. Critical reviews and customer complaints forced compiler and interpreter makers to modernize BASIC and create more powerful, structured versions of the popular language. These revised editions included True BASIC (1985), Microsoft QuickBASIC (1985), Borland Turbo Basic (1987), and Microsoft BASIC Professional Development System (1989). (See Figure 5.7 for one of the most popular structured BASIC reference manuals of the era.) The feature enhancements in these products included user-defined subprograms and functions, the ability to create local

Ahmed, "Students' Thought Processes While Engaged in Computer Programming" (Ph.D. diss., Oregon State University, 1992).

21. John Clark Craig, *Microsoft Visual Basic Workshop, Windows Edition* (Redmond, WA: Microsoft Press, 1993), 4.

22. Craig, *Microsoft Visual Basic Workshop*, 5.

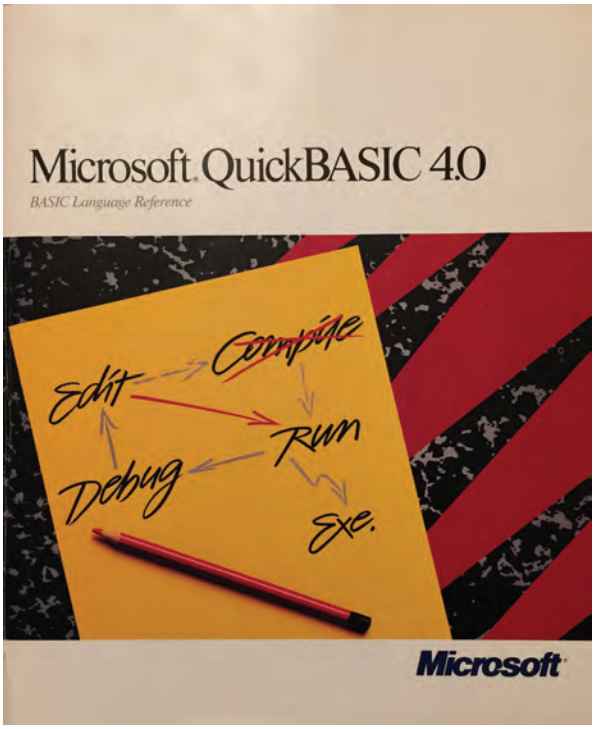


Figure 5.7 **Microsoft QuickBASIC 4.0 Language Reference (1987).** (Used with permission from Microsoft)

variables and constants in procedures, and the integration of popular control flow structures such as Do... While and Select... Case.

In most of the new versions, line numbers and labels became optional in code, but they were allowed for backward compatibility with earlier products. Finally, in graphical operating systems like those supplied with the Apple Macintosh and Commodore Amiga computers, the designers integrated event-driven programming features into their language projects, including the ability to create menus, dialog boxes, text-entry fields, buttons, and multitasking controls. These features would later become hallmarks of Visual Basic for Windows and Visual Basic for MS-DOS in the early 1990s.

5.6 Microsoft Press and *Learn BASIC Now*

To highlight one way that structured BASIC helped students and self-taught programmers to construct their programs, we'll examine the origins of the Microsoft Press book *Learn BASIC Now* (1989), a QuickBASIC primer that I

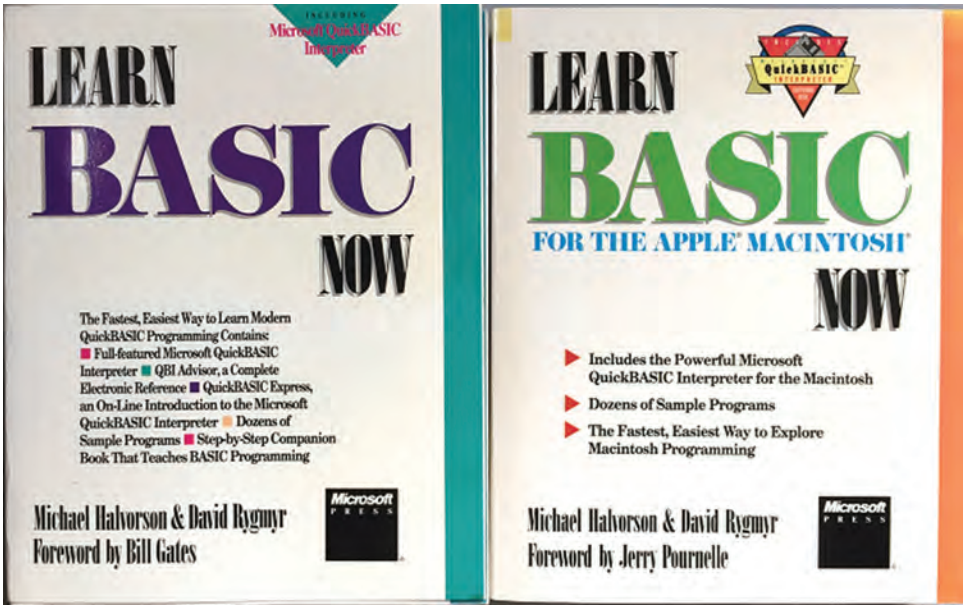


Figure 5.8 ***Learn BASIC Now*, by Michael Halvorson and David Rygmyr, published in IBM PC and Apple Macintosh editions (1989 and 1990). Both books were designed for students and self-taught programmers, and featured bundled versions of the Microsoft QuickBASIC Interpreter.** (Used with permission from Microsoft)

co-authored with David Rygmyr while working for the book publishing division of Microsoft Corporation. (See Figure 5.8.) This book provides a convenient vantage point for our analysis of BASIC programming culture in the 1980s and 1990s, because it was one of the first products to combine a course of programming instruction with a structured BASIC software product, the Microsoft QuickBASIC Interpreter.[23] *Learn BASIC Now* was published in 1989, near the high-point of BASIC saturation in the PC marketplace. During that year, the installed base of BASIC and QuickBASIC programmers hit approximately four million active users in the U.S., an indication that BASIC programming had become a dominant force on the IBM PC / MS-DOS platform.

In preparing *Learn BASIC Now*, the Microsoft Press editorial team directly appealed to the computer literacy strategies discussed in the last two chapters. They prioritized accessible language for instruction, offered interactive learning modules for new users, included enticing illustrations, offered review questions

23. Michael Halvorson and David Rygymr, *Learn BASIC Now* (Redmond, WA: Microsoft Press, 1989). For the Macintosh edition, see Michael Halvorson and David Rygmyr, *Learn BASIC for the Apple Macintosh Now* (Redmond, WA: Microsoft Press, 1990).

and exercises for students, and sold the book as part of an integrated book-and-software learning package. *Learn BASIC Now* also presented itself as a modern, state-of-the-art introduction to structured BASIC using contemporary computer science terminology, avoiding the perceived pitfalls of earlier dialects and approaches.

The book project was initiated in 1988 and involved two staff authors (Halvorson and Rygmyr), two staff editors (Megan Sheppard and Dail Magee, Jr.), and an experienced team of in-house artists, proofreaders, compositors, and support specialists. The book publishing division's acquisitions editors, marketers, sales personnel, and executives also took a sustaining interest in the project. At the time, Microsoft Press employed approximately 65 fulltime staff members at its Redmond, Washington campus, where they were considered regular employees of Microsoft Corporation. By the end of 1988, there were 2,087 employees at Microsoft throughout the organization, including the staff members of Microsoft Press.[24]

As part of the team's planning efforts, they sought and received permission to bind a copy of the Microsoft QuickBASIC Interpreter into the book, which was stored on three 5.25" (360KB) floppy disks. The first book was designed for IBM PCs and compatibles running MS-DOS versions 2.1 and higher. (The current version of DOS at the book's release was 4.0.) In 1990, we published a Macintosh version of *Learn BASIC Now*, which included the Microsoft QuickBASIC Interpreter for Macintosh Plus, SE, and II systems on 3.5" diskettes.

The first volume included a Foreword by Bill Gates, one of the co-developers of Altair BASIC. Gates struck a populist tone in his introduction. He argued that continued innovation in the computer industry would require more than Computer Science graduates, it would require all people. "We need a diverse community of users creating tools and solving problems to fully achieve the potential of the microcomputer," Gates wrote.[25]

The Macintosh version began with a Foreword by *Byte* columnist and science fiction writer Jerry Pournelle. He opened as a contrarian, a tact he often took in his *Byte* columns. "Some computer fanatics laugh at BASIC. Real Programmers, they say, write in Assembler, or Forth, or C, or APL, or even Pascal. Never BASIC, though." After acknowledging the jibes of engineers and pundits, however, Pournelle connected BASIC to the Macintosh in a creative way:

> True, modern compiling BASIC, the QuickBASIC this book does such an admirable job of teaching, is a lot different from the BASIC I first learned—but then so are the machines we use now. There's a strong parallel

24. Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: The MIT Press, 2003), 233.

25. Bill Gates, "Foreword," in Halvorson and Rygmyr, *Learn BASIC Now*, x.

> between the development of BASIC and the development of the Macintosh computer. Each was, when introduced, the most learnable product of its class: With both early Microsoft BASIC and the first 128K Mac, you could sit down and do something interesting within a few minutes.[26]

Pournelle emphasized the special relationship between BASIC and the Mac, a pairing that was often neglected. A consummate futurist, Pournelle also made a prediction: "I don't know how long it will be before the ability to program—or at least to understand what programming is about even if you don't actively do it—will be a requirement for a significant number of jobs. I suspect that day is coming more quickly than most realize. Anyway, you've got the right tools for the first step."[27]

In a sense, both Gates and Pournelle were interpreting the learn-to-program movement through the lens of earlier mythologies about programming and computers. But they also perceived a future for the movement outside of educational circles. BASIC programming could serve as a vehicle for self-fulfillment, connecting with other users, and finding a job.

Rygmyr and I were mindful of the movement's history as we constructed the book. The title of our product was a deliberate play on Ted Nelson's 1974 mandate, "You can and must understand computers NOW." We certainly believed that Nelson's ideals were relevant, and his call to arms still seemed rousing for contemporary readers. In 1987, Microsoft Press had republished Nelson's *Computer Lib/Dream Machines* with information about PCs and a renewal of the call for citizens to embrace the information age. (See Figure 2.7.) The audience for *Learn BASIC Now* was thus specifically intended to be self-taught programmers and hobbyists with little or no coding experience. It was a book for average Americans who wanted to learn the fundamentals of programming outside of academic contexts.

The structured QuickBASIC Interpreter was the component that made *Learn BASIC Now* unique, and we found room on the companion disks to include all the sample code that we developed in our projects. The text-based IDE included mouse support, drop-down menus, QuickBASIC documentation, and helpful editing features, making the software a major upgrade over the GW-BASIC software that was currently shipping with MS-DOS. Using the enhanced IDE, readers could type in, test, and debug their programs almost instantly. The interpreter was based on QuickBASIC version 4.5 (released in 1988), a compiler that made procedural programming and structured design easier, and we took full advantage of the upgrades. In the entire 490-page book, we never used a GoTo statement in our code, making

26. Jerry Pournelle, "Foreword," in Halvorson and Rygmyr, *Learn BASIC for the Apple Macintosh Now*, ix.

27. Pournelle, "Foreword," in Halvorson and Rygmyr, *Learn BASIC for the Apple Macintosh Now*, xi.

the point that academic and professional best-practices could be introduced *from the beginning* to our intended audiences. The major topics in the book included an introduction to problem solving, building algorithms, using variables and operators, controlling program flow, working with loops, creating and calling subprograms and functions, working with arrays, string-processing techniques, using files and databases, integrating graphics and sound, and debugging.

A profile of the teams that worked on *Learn BASIC Now* will demonstrate the collaborative nature of computer book publishing in the 1980s, a unique branch of the publishing world that drew contributors from a variety of professional backgrounds. Microsoft Press played a unique role in the PC industry. Since its founding in 1983, the book publisher sought to enhance the experience of computer users by publishing the works of independent authors about popular applications and programming systems. Microsoft Press did not produce the technical documentation for Microsoft's products; this important task was embedded within the company's product groups. Instead, Microsoft Press distributed publications through traditional trade and education book channels, both in the North American markets and through international publishing partners. They published a full range of products about the evolving PC industry, including books about products from Microsoft, Apple, Ashton-Tate, Lotus Development, Santa Cruz Operation, and other companies. Microsoft Press also produced general works about science and computing, including *CD-ROM: The New Papyrus* (1986), *Programmers at Work* (1986), *The World of Mathematics* (4-vols., 1988), and *The Microsoft Press Computer Dictionary* (1991). One of their strengths was an editorial team that had depth in both technical and trade publishing, with connections to the publishing hotbeds of New York, San Francisco, and the Midwest. (See Figure 5.9.)

The writing and editorial group that worked on *Learn Basic Now* had a diverse range of professional experiences. Co-author David Rygmyr (1958–) was especially fascinated with electronics, puzzles, and sound effects. (See Figure 5.10.) A true tinkerer, Rygmyr was interested in programming and classic computer games, and he maintained an impressive collection of PC-gaming titles, often setting them up for the technical editors to play. Rygmyr did his first professional work with computers in the late 1970s by preparing paper tape and punched-cards for a UNIVAC mainframe. He joined Microsoft Press in 1984 and worked as a technical manager and senior technical editor, supporting the editorial teams when complex challenges arose involving new hardware, networking, DOS internals, C programming, and more.

Megan E. Sheppard (1959–) brought her considerable editorial and management expertise to the *Learn BASIC Now* project. Sheppard started work at Microsoft in 1983 and settled into senior roles as an editor and program manager at Microsoft

Figure 5.9 **Members of the Microsoft Press editorial group on the Redmond campus in late 1986. Shown here are a selection of the organization's manuscript editors, technical editors, proofreaders, word processors, and editorial managers. (Halvorson is left, seated, with arms extended; Rygmyr is in the back row, center, with light-colored hair.)** (Used with permission from Microsoft)

Press. Sheppard is "an editor's editor," always determined to transmit clear, accurate, and direct prose. She often took the lead role in determining how a publishing project should be organized, scheduled, and produced. Sheppard worked over the years with Peter Rinearson, Charles Petzold, and Dan Gookin—all best-selling authors who wrote with passion for the power user and do-it-yourself segments of the PC marketplace.

The team was also supported by Dail Magee, Jr., a skilled technical editor who tested every program, verified the manuscript's integrity, and served as an important conduit between the authors and Microsoft's product teams. Magee joined Microsoft Press in 1988, after studying Electronics and Computer Science at Walla Walla College. This book was one of the first that Magee worked on at Microsoft Press, and he went on to have a long career in publishing and software development that continues as this book goes to press. After our two *Learn BASIC Now* titles were published, Magee also handled product support for the books. At that time, product support entailed answering letters and phone calls from customers who had purchased products and needed more information. In fact, it was not unusual for the Microsoft Press editorial staff to call readers personally if they wrote in with

Figure 5.10 **The core *Learn BASIC Now* product team. Left to right: Dale Magee, Jr., Megan Sheppard, David Rygmyr, and Michael Halvorson (1990).** (Photo courtesy of Megan Sheppard)

concerns. In the 1980s, virtually any letter that arrived from a reader was given special significance. By the early 1990s, however, support for most products was encapsulated in Microsoft Knowledgebase articles, which were posted in online databases so that any Microsoft employee handling support inquiries could access the information.

How did we decide the style and format used for our programming course? In the 1980s, the method that experienced programmers used for learning a new language still entailed working through a printed language specification such as *The Pascal User Manual and Report* (1971), by Kathleen Jensen and Niklaus Wirth, or *The C Programming Language* (1978), by Brian Kernighan and Dennis Ritchie. These

short books were *de facto* language standards written by the original designers of the compilers, and they were highly esteemed. However, we were more deeply influenced by the primer *Oh! Pascal!* (1982), written by Doug Cooper and Michael Clancy. This book for the educational marketplace took a very light-hearted approach to building algorithms and learning the Pascal language.[28] For example, there was a picture of Leonard Nimoy (the character Spock from Star Trek) in the Boolean expressions section of the book, and there was a genealogical chart showing the ancestry of Greek gods as a way of introducing "tree" data structures. After some discussion, we decided to create a language primer that was light-hearted, fun to use, and culturally relevant, replete with well-organized sample code and illustrations that would keep the attention of readers.

I remember having an extended conversation about *Oh! Pascal!* with Patty Stonesifer (1956–), a publishing executive from Que Corporation who joined Microsoft Press in 1988 as Director of Sales. (See Figure 5.11.) Que was an Indianapolis-based computer book publisher known for its "Using" series. Most Que books were published with all-black covers and white lettering, which made the titles easy to recognize on bookstore shelves. Stonesifer had a technical writing background herself, and she encouraged the *Learn BASIC Now* team to make our primer friendly and quirky, arranging for Bill Gates to write a foreword to the book because of his long association with BASIC. Stonesifer's individual attention to our project was typical of management generally at Microsoft Press—a tradition cultivated in the early years by Nahim Stiskin, Min Yee, Susan Lammers, Jim Brown, and Elton Welke. Stonesifer was soon transferred to prominent positions in Microsoft International and Microsoft Canada, with later leadership roles in Microsoft Product Support and the Consumer Products group. She eventually became a senior vice president of Microsoft Corporation and later the CEO of the Bill and Melinda Gates Foundation.[29]

After numerous editorial passes and months of work, *Learn BASIC Now* was ready for publication. Benefiting from the rising interest in BASIC, *Learn BASIC Now* sold over 75,000 copies in its first edition. In its review of the book, the *New York Times* wrote, "For anyone who wants to learn something about programming, it would be hard to find an easier or more cost-effective source than *Learn BASIC Now*."[30] New programmers enjoyed the book's approach and told us so, but the

28. Doug Cooper and Michael Clancy, *Oh! Pascal!* (New York: W.W. Norton & Company, 1982).

29. For a summary of Stonesifer's career and recent projects, see June Thomas, "The retreat observation that put Patty Stonesifer on Microsoft's leadership radar," *Slate*, March 28, 2019. https://slate.com/business/2019/03/patty-stonesifer-microsoft-marthas-table-transcript.html. Accessed on August 5, 2019.

30. Halvorson and Rygmyr, *Learn BASIC Now*, back cover.

Figure 5.11 **Patty Stonesifer became Director of Sales at Microsoft Press in 1988 and soon took leadership roles in other divisions of Microsoft. Her career was built on many years of experience in the technical publishing industry.** (Used with permission from Microsoft and Patty Stonesifer)

product also benefited from the powerful QuickBASIC Interpreter that was distributed with the integrated book-and-software package. Microsoft sold the QuickBASIC version 4.5 Compiler for $99 at the time, and the QuickBASIC Interpreter was priced at $39.95. Our learning product was also priced at $39.95, but it included the interpreter software, our 490-page companion book, disks with sample code, and a full IDE with an electronic help system and reference. The comprehensive package was clearly a good value for self-taught programmers who wanted an integrated learning product to get started.

5.7 Microsoft Game Shop

In typical fashion, Microsoft Press continued with some follow-on projects, sensing the popular demand for programming language products. In 1990, a Macintosh edition of *Learn BASIC Now* was published (discussed already), complete with a Macintosh version of the QuickBASIC Interpreter. Microsoft Corporation also became interested in the BASIC game programming market, and they released a

special edition of our book-and-software package for the MS-DOS gaming audience. This product was entitled *Microsoft Game Shop: Games and the QBasic Learning Environment*.[31] (See Figure 5.12.) The $49.95 software package debuted in 1990, and it included the MS-DOS version of the *Learn BASIC Now* book, the QuickBASIC Interpreter, the IDE and help system, and a selection of six "arcade-style" PC games written in QBasic. The featured program in the collection was *QBlocks*, a Tetris-style BASIC game that programmers could modify to learn more about coding and gaming on IBM PCs and compatibles. An advantage of the book-and-software package was that *QBlocks* was provided on disk—it did not need to be typed in from the pages of a book or magazine as the first BASIC games from this era had been. Despite the length of the newer QBasic programs (which were organized into structured modules), there was still a thematic connection to earlier BASIC programming adventures like David Ahl's *101 BASIC Computer Games*.

Compute! magazine published a thorough review of the QBASIC gaming tutorial, emphasizing the continuing relevance of tinkering with games to learn coding skills. The article emphasized the customizable features of the *QSpace* program (a variation of *Missile Command*), which could be edited to modify the game's sound effects, missile speed, explosions, and colors used for graphics.[32] The book-and-software package was depicted as a timely vehicle for "creative recreation," though which users could learn programming skills while having fun—just like in 'the old days.'[33] The reviewer offered some perspective on the recent computer literacy debates:

> Years ago, being computer literate meant learning to program in BASIC. Although there is less emphasis on programming today, programming remains a challenging, creative, and even entertaining enterprise; and computer users who understand programming will always be a step ahead of those who don't.
>
> BASIC is an ideal first language. Its ease of use provides programmers with a simple way to learn and experiment with programming concepts that are the building blocks of programs written in any language.
>
> For learning to program, *Microsoft Game Shop* provides an excellent introduction with plenty of fun and useful tools. First time programmers have a

31. For a copy of this software and its marketing materials, see *Microsoft Game Shop*, Microsoft Corporation (1990), Computer History Museum, Fremont, CA, cat. no. 102778084.

32. Tony Roberts, "Microsoft Game Shop," *Compute!*, Issue 130 (June 1991), 136–139, here at 138.

33. Roberts, "Microsoft Game Shop," 136, 139. Note that at the time "the old days" would have been just 10 years (or less) in the past.

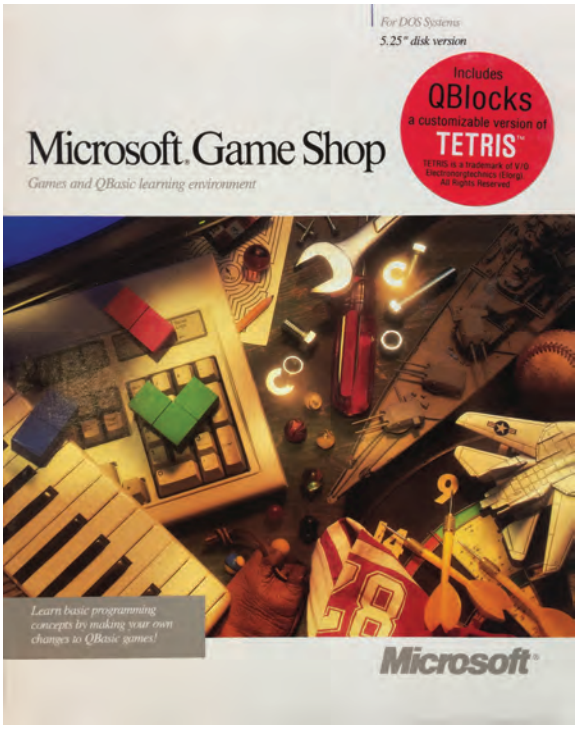


Figure 5.12 **Microsoft Game Shop, a PC gaming package with a built-in programming tutorial for IBM PCs and compatibles (1990).** (Used with permission from Microsoft)

chance to experience the challenge and excitement of the old days of computing, but with all the comforts afforded by today's technology.[34]

You might have noticed the term "QBasic" in the product labeling above as a short-hand reference for the QuickBASIC programming language. When MS-DOS 5 was released in 1991, the company decided to make this product name official. MS-DOS 5 included the structured QuickBASIC Interpreter as a standard feature (replacing GW-BASIC), but the interpreter was renamed QBasic to avoid confusion with the full-featured QuickBASIC Compiler that still sold as a retail product. Now that Microsoft was including a structured, procedural version of BASIC with MS-DOS, book publishers could update all of their BASICA and GW-BASIC books to emphasize structured programming concepts. At Microsoft Press, the *Learn BASIC Now* team reissued our book for the MS-DOS platform as *Running MS-DOS QBasic* (1991), a product that sold for the reduced price of $22.95 because it did not require

34. Roberts, "Microsoft Game Shop," 139.

an expensive set of companion disks for the interpreter, which was now included in most versions of MS-DOS.[35] The book went through six printings and sold 75,000 copies in the first 2 years, indicating that the market for structured BASIC (and QBasic) continued to be strong under MS-DOS versions 5 and 6.

Clearly the arrival of many commercial PC applications had obviated some of the need for users to build their own programs in BASIC. However, the release of new "structured" compilers and interpreters gave a boost to modernized BASIC in the hobbyist and non-professional programming markets. Computer gaming also presented some exciting opportunities for BASIC programmers. The PC industry had cut its teeth on this "beginner's" language in the 1970s and by the early 1990s there was still some vitality left in the coding technology and its substantial user base.

5.8 Visual Basic for Windows

In 1991, BASIC programmers encountered a new programming paradigm as Microsoft released Microsoft Visual Basic for Windows 1.0, a RAD tool capable of creating Windows applications in considerably less time than the process took using traditional coding methods. In early 1991, the conventional process for creating a Windows application begin with implementing the core program logic using a C compiler from either Microsoft or Borland International. The application then needed to conform to the many requirements of the graphical, multitasking operating system. These included preparing visual elements and using the various application programming interfaces (APIs), libraries, and component tools in the Microsoft Windows Software Development Kit. I describe this process in more detail in Chapter 10, which explores the origins of the C programming language and the compiler's use under the Unix, MS-DOS, Macintosh, and Windows operating systems.

Conceptually, Microsoft Visual Basic was the result of a synthesis between Alan Cooper's work on "Ruby," a RAD tool for Windows, and the Microsoft QuickBASIC product, which was revised and enhanced to support programming under Windows.[36] The exciting thing about "Ruby" for developers (code-named "Thunder" at Microsoft) was that it allowed users to construct their program's

35. Michael Halvorson and David Rygmyr, *Running MS-DOS QBasic* (Redmond, WA: Microsoft Press, 1991).

36. "Ruby" was the code name for a visual programming language (*c*. 1988) that ran under Windows. Alan Cooper's "Ruby" has no specific relation to the modern Ruby programming language, developed by Yukihiro Matsumoto in Japan.

graphical user interface (GUI) by visually arranging software components on a form using tools from a familiar palette, called the Toolbox.

In the final version of Visual Basic 1.0, developers could create windows, dialog boxes, menus, scroll bars, buttons, and other program elements using simple drag-and-drop techniques. Once they had placed the visual elements of their application on a form, they could specify *properties* for the components using point-and-click settings and coded *event procedures*. The properties control how the components appeared and functioned when the program ran. The coded event procedures were created in modules that resembled structured BASIC routines. In some cases, code snippets from earlier languages like QuickBASIC could be reused.

A fundamental change to the programming paradigm was that Visual Basic applications were object based. Rather than executing from the top of a program to the bottom in linear fashion, object-oriented systems like Visual Basic presented collections of "intelligent" objects before the user in an application window, which the user could then manipulate and use. This style of programming is also referred to as "event-driven programming," because the job of the programmer is to write code that responds to the various *events* that happen when the user clicks or drags on items in the GUI. One of the exciting advancements in Visual Basic was that so much happened automatically when it came to designing the user interface. When the programmer dragged objects from the Toolbox to the application form, the objects already "knew" how to function in a basic sense. For example, a button object "knew" its basic shape and operating context, including what to do if it was clicked or double clicked. Taking this functionality as a starting point, the Visual Basic programmer could then customize a button by placing VB code in an event procedure associated with the button's Click event. From a programmer's point of view, there were two "worlds" of Visual Basic development—the world of visible objects created on a form using the Toolbox and the Properties windows, and the world of VB code in event procedures and modules that lay "beneath" the objects and forms.

Microsoft Visual Basic 1.0 for Windows was released in May 1991 at the Windows World trade show in Atlanta, Georgia. (For more about the commercial context of these shows, see Chapter 11.) Mindful of the importance of the MS-DOS platform, Microsoft also released a DOS-based version of Visual Basic in September 1992, named Microsoft Visual Basic for MS-DOS 1.0. Both products supported the event-driven programming model, although they were not fully compatible. (Visual Basic for MS-DOS is best understood as an extension of the QuickBASIC and BASIC Professional Development System product lines.) Visual Basic's impact was further extended by the introduction of Visual Basic for Applications (VBA) in numerous commercial Windows applications, including Microsoft Excel, Microsoft Word,

and Microsoft Access. (The technology debuted with Microsoft Excel 5.0 in 1993.) Describing VBA as a "unifying language," Microsoft promoted the technology by suggesting that it allowed users to write user-defined functions within their applications that processed data programmatically and allowed communication between applications.

I was acquiring new books for Microsoft Press during the release of these products, and I remember meeting with Nevet Basker and Adam Rauch in the Microsoft Languages group to see a demo of "Thunder," the initial test version of the Visual Basic for Windows compiler. Nevet Basker was the company's first product manager for Visual Basic, and she was relatively new to Microsoft herself, arriving in the Summer of 1989 as a student intern. She went on to have a productive career at the company, serving as the product manager for Microsoft Access 1.0 and Microsoft FoxPro 2.0. Adam Rauch was the program manager for Visual Basic, overseeing the product's engineering specification and keeping the development team on track during the product's unique construction process. The Visual Basic for Windows 1.0 team at Microsoft also included Nancy Barnes, Scott Ferguson, John Fine, Chris Fraley, Brian Lewis, and Rick Olson. Back at Microsoft Press (in a neighboring building on the Microsoft campus), we went to work developing titles for what looked to be an intriguing line of products.

In the beginning, it seemed that Microsoft had relatively modest expectations for the new Visual Basic product line. Visual Basic was released at Windows World (a trade show combined with COMDEX/Spring '91), and the Languages marketing group seemed as surprised as the rest of us when Visual Basic for Windows quickly became the dominant development tool in the Windows marketplace. Although Visual Basic did not surpass the Microsoft C Compiler as the major *commercial* tool for software development on the Windows platform, Visual Basic was used by millions of aspiring software developers who hoped to learn Windows programming quickly and turn their ideas into games, utilities, and simple business applications. Visual Basic was advertised as a RAD tool that could help users to build graphical, drag-and-drop-style applications fast, including database front ends, graphics-based video games, popup menus, system accessories, and so on. The academics and professional developers who held lingering stereotypes about older versions of BASIC were encouraged to set aside their biases and give Visual Basic a try. To emphasize this point, Microsoft Group Product Manager Tom Button defended the compiler in 1992 in *Computerworld* magazine. "The source of BASIC's [tawdry] reputation is [from] decades of old interpreted technology, not on compiled event-driven Basic."[37] In the article, he pointed out that 113,000 business applications

37. Garry Ray, "Basic gains commercial respect," *Computerworld*, November 16, 1992, 128.

had already been created in Visual Basic just a year after the product's first release, and they were thriving in corporate America.[38]

5.9 Innovative Programming Primers

Microsoft Visual Basic gained in popularity and became a globally successful product, used by millions of programmers to write their first Windows applications. But how did students and self-taught learners adopt these new coding skills? How did this new technology impact America's learn-to-program movement?

By the time Visual Basic for Windows was released in 1991, the Windows platform had also grown and matured, and there were numerous book publishers vying for opportunities to document it.[39] These included Microsoft Press, IDG Books, Howard W. Sams, O'Reilly, Osborne McGraw-Hill, Que, Sybex, Ventana Press, Wiley, Wrox, and Ziff Davis Press. Although few of these companies were ready with books when the VB 1.0 release took place, the publishing industry gradually caught up with demand by Microsoft Visual Basic 3.0, which was released to customers during the summer of 1993. By that point, computer book and magazine publishers could market their products to a Visual Basic user base that numbered in the millions and included programmers with a variety of skill levels and interests. (For a selection of Visual Basic programming books, see Figure 5.13.) For a time, Visual Basic became *the* vehicle to introduce programming concepts to users of the Windows platform, the largest sector of the PC marketplace.

Among the first resources for new Visual Basic programmers was Ross Nelson's *Running Visual Basic for Windows* (1992), released for Microsoft Visual Basic 2.0 and soon updated for the Visual Basic 3.0 product.[40] Nelson had worked at Intel Corporation on the Intel 80286 and 80386 microprocessors, and he was a regular columnist for *Byte* and *Dr. Dobb's Journal*. At home as a teacher of assembly language skills and techniques, Nelson was known primarily for his work on a successful assembly language primer, *The 80386 Book* (1988), which taught software developers how to make the most of Intel's speedy new microprocessor.[41] But Nelson also became a convert to Visual Basic programming soon after the

38. Ray, "Basic gains commercial respect," *Computerworld*, 128.

39. In the 1990s, the PC/Windows platform was also referred to as the "Wintel platform" because of its fruitful combination of Intel microprocessors and the Microsoft Windows operating system. For a discussion of the subtle differences among PC platforms and the challenges in labeling them, see James Sumner, "What makes a PC? Thoughts on computing platforms, standards, and compatibility," *IEEE Annals of the History of Computing* 29, no. 2 (2007): 88–89.

40. Ross Nelson, *Running Visual Basic for Windows: A Hands-on Introduction to Programming for Windows*, Second Edition (Redmond, WA: Microsoft Press, 1993).

41. Ross Nelson, *80386: The 80386 Book* (Redmond, WA, Microsoft Press, 1988).

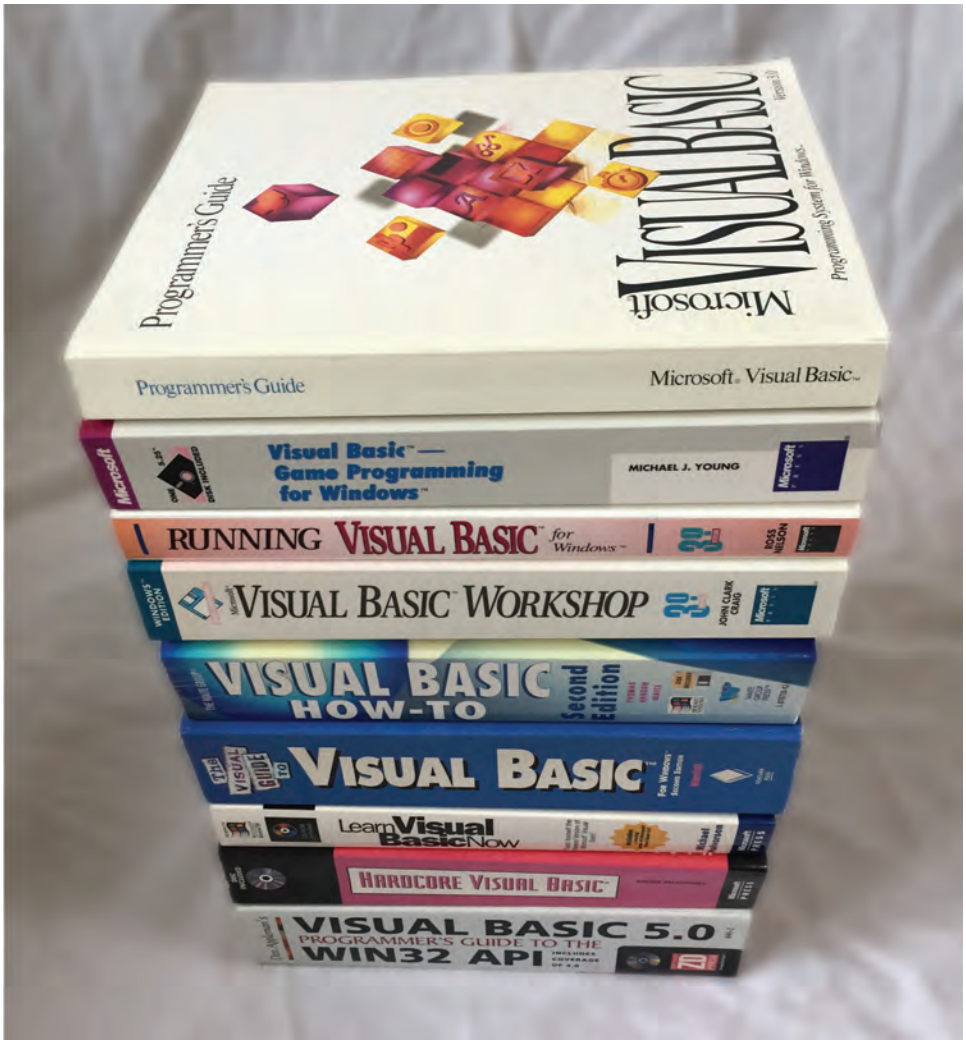


Figure 5.13 **A selection of popular Microsoft Visual Basic programming books published in the early 1990s after the first releases of Microsoft Visual Basic for Windows. The publishers represented include Microsoft Press, Ventana Press, Waite Group Press, and Ziff-Davis Press. At the top of the stack is the Microsoft Visual Basic 3.0 Programmer's Guide, included with the product.** (Photo courtesy of Michael Halvorson. Microsoft products used with permission from Microsoft)

product's release. In *Running Visual Basic*, he emphasized how easy it was for newcomers to learn the language and tools, even if they had no programming experience. However, a little mathematics background wouldn't hurt:

> You don't need to know a lot of computer buzzwords to use this book, nor do you need to have programming experience… It will be helpful if you have created macros for your spreadsheet or word processing program. And, finally, you shouldn't be afraid of a little algebra. This book contains no heavy math, but remember that computer languages were originally designed for performing mathematical computation; consequently, a little of that heritage remains in every computer language today.[42]

Running Visual Basic focused on the VB coding language, which was closely related to QuickBASIC. The steps Nelson recommended for building the user interface were covered in just two chapters. (This was a limitation, or shortcoming, of most early Visual Basic programming books). In fact, it seemed to take years for most programmers (and authors) to understand how truly daunting it was to create a user interface from scratch that could be used effectively by a range of users, even when the software developers had impressive design tools. An obvious consequence of this problem was that the first Windows applications created in Visual Basic often presented non-standard menus, dialog boxes, and commands to the user, who found the lack of consistency frustrating. In many respects, the nascent field of user experience (UX) design was still in its infancy.[43]

A more intensive introduction to Visual Basic programming techniques was provided by John Clark Craig, an experienced author and engineer who relished in the "learn by doing" approach when he constructed a comprehensive book-and-software bundle containing hundreds of pages of source code. In *Microsoft Visual Basic Programmer's Workshop* (1991), published by Microsoft Press, Craig presented imaginative code modules and user forms that programmers could integrate into Windows-based applications. The first edition of the *Workshop* was designed for Visual Basic 1.0, and the revised edition, published in 1993, included reworked examples for Visual Basic versions 1.0 through 3.0.[44] The Visual Basic 3.0 upgrade was designed for the Windows 3.1 operating system, an enhancement of the Windows platform that gained considerable traction in the PC marketplace. (For more about the reception of Windows 3.0 and 3.1, see Chapters 6 and 10.) Essentially, Visual Basic 3.0 offered support for Access databases via the new Visual Basic Data control, object linking and embedding (OLE) integration, dynamic data exchange

42. Nelson, *Running Visual Basic for Windows*, xviii.

43. To address the situation, Microsoft soon published specific guidelines for the Windows 95 operating system entitled *The Windows Interface Guidelines for Software Design* (Redmond, WA: Microsoft Press, 1995).

44. John Clark Craig, *Microsoft Visual Basic Workshop, Windows Edition* (Redmond, WA: Microsoft Press, 1993).

(DDE), and the capability to distribute applications via the SetupWizard. Craig's code samples and utilities took advantage of these new features and more, and his technical writing had great appeal to scientists and tinkerers. Craig's sample programs included working screen savers, random number generators, custom controls, file compression utilities, and a demo that used a modem and an atomic clock in Denver, Colorado to set a PC's system time—down to the millisecond.[45]

Game programmers on the PC platform were treated to a creative primer on programming graphics, sound, and animation with *Visual Basic—Game Programming for Windows*, by Michael J. Young (1992).[46] This computer book provided 12 ready to run Windows games and fractals for Visual Basic 1.0 programmers to experiment with and customize. The text provided a short introduction to Windows programming in Visual Basic, and then Young launched into a clever selection of puzzles, strategic board games, and action-gambling games. Like Dian Crayne, Michael Young discussed strategy and game design principles in his book, but the sections on graphics and animation demonstrated the power of Visual Basic to create visual effects under Windows. The games were included on a bound-in companion disk, so that readers could avoid the drudgery of typing the programs in manually. I served as the acquisitions editor for this project, which took shape after a conversation with Michael Young about how rapidly game programming had evolved on the PC platform. I was so impressed with Young's work that we remained in close contact and eventually co-authored a series of power-user books about Microsoft Office, the Windows application suite that debuted as an integrated bundle in 1995.[47]

In late 1993, I left Microsoft Corporation and began writing computer books fulltime. I also entered a doctoral program in History at the University of Washington in Seattle, where I studied European history and the history of technology. One of my first projects as an independent author was to work on a series of Visual Basic programming primers entitled *Microsoft Visual Basic Step by Step*, first published for the users of the Microsoft Visual Basic 4.0 Development System, which debuted in

45. The system time application used the new Communications control provided by Microsoft Visual Basic 3.0 Professional Edition. It dialed an atomic clock at the National Institute of Standards and Technology and downloaded the current date and time. Time checks like this were necessary before computers were connected to the Internet and generally able to adjust system settings automatically. See Craig, *Visual Basic Workshop*, 428–434.

46. Michael J. Young, *Visual Basic—Game Programming for Windows* (Redmond, WA: Microsoft Press, 1992).

47. Michael Halvorson and Michael Young, *Running Microsoft Office for Windows 95* (Redmond, WA: Microsoft Press, 1995). The first edition of our book came in at 1064 pages. Under intense time and page count pressures, Young seemingly had no problem meeting this or other writing deadlines.

August 1995.[48] These books attempted to more fully introduce Windows design principles and graphical programming considerations into a programming course than competing titles in the marketplace, while still giving due consideration to language instruction, algorithms, and classic topics such as debugging computer software. I also revised *Learn BASIC Now* into a book-and-software package that taught Visual Basic programming skills in the context of the Windows platform. *Learn Visual Basic Now* included a "working model" of the Visual Basic software, which newcomers could use to build working Visual Basic applications, even without the full retail release of Microsoft Visual Basic. (Readers could create their programs, run them in memory, and share them with others, but they could not create executable files or distribute applications until they purchased the full retail edition.) Microsoft eventually established academic pricing for software like this through a program called Microsoft DreamSpark, which offered programming software to college students and faculty at deep discounts.

I published a Visual Basic book in the Microsoft Press *Step by Step* series for each version of Visual Basic released between 1995 and 2013 (10 editions in all). The Visual Basic software enjoyed strong commercial success during those years and so did my *Step by Step* books, which were occasionally bestsellers in the Programming Languages and Visual Basic/Visual Studio publishing categories tracked by *O'Reilly Media* and *Publisher's Weekly*. My Visual Basic primers eventually sold over 600,000 copies collectively, contributing to the continuation of the learn-to-program movement on the Windows platform as it expanded into new commercial and graphical programming contexts.

By the late 1990s, Visual Basic and most commercial development systems were being revised every 18 to 24 months by software publishers, who typically aligned their product cycles with the new operating system releases for Windows, Windows NT, the Macintosh, and related systems. Computer book publishers were very dependent on these schedules, and they timed their primers and reference guides so that they would be announced and sold at the same time as the commercial products. For those learning to program in these years, it is likely that they learned in the context of specific software products and schedules, and aspiring developers would have been encouraged to upgrade their compilers and development systems along with everyone else in the PC software industry. This new direction of the learn-to-program movement I associate with the corporate and commercial contexts of the PC software industry as it approached maturity in the mid-1990s. I will have much more to say about these commercial developments and their consequences in Parts II and III of this book.

48. Michael Halvorson, *Microsoft Visual Basic 4 for Windows 95 Step by Step* (Redmond, WA: Microsoft Press, 1995).

There were also *advanced users* of Visual Basic who sought to create commercial-grade applications and utilities for the Windows marketplace, particularly after the release of Visual Basic 4.0, which included a raft of new features to support object-oriented programming and use of the Windows API. A new wave of experienced authors and developers wrote books for these audiences. Bruce McKinney's *Hardcore Visual Basic: Secrets, Shortcuts, and Solutions for Programming Windows without Using C* (1995) communicated a deep appreciation for the design of Visual Basic, but quickly went beyond the standard features to teach advanced techniques that Windows developers usually implemented in the C language. In particular, McKinney taught Visual Basic programmers to use "pointers" (references to memory locations), so that they could make effective use of the Windows API. Dan Appleman's massive *Visual Basic 5.0 Programmer's Guide to the Win32 API* (1997) built on this exploration of Windows internals by publishing reference materials for the vast collection of Windows APIs that were accessible to Windows 95 and Windows NT 4.0 applications. This allowed Visual Basic 4.0 and Visual Basic 5.0 users to implement features that were typically reserved for experts who were using Microsoft C and the Windows Software Development Kit to create applications.

In summary, Microsoft Visual Basic became successful enough in the PC marketplace to attract users with a wide range of programming proficiencies—from hobbyists and students to gamers and professional developers. By the mid-1990s, computer book and magazine publishers presented a range of learning products and commentary for these users, anticipating the "scaffolded" approach to training that would sustain the learn-to-program movement through the era of enterprise computing.

How did the transition from structured BASIC to Visual Basic impact the installed base of BASIC programmers? By 1999, Microsoft announced that Visual Basic had sold over 3.5 million copies, bringing Visual Basic to roughly the same level of market penetration that structured BASIC and QuickBASIC had enjoyed in the late 1980s. Computer books related to Visual Basic were among the top-selling programming primers in the U.S., filling the shelves next to popular programming books on Turbo Pascal, C, C++, and assembly language. For the users that remained loyal to the MS-DOS platform (rejecting Windows and the Macintosh), QuickBASIC, QBasic, and Borland Turbo Basic remained the leading programming tools.

The long arc that spanned from time-sharing BASIC to microcomputer BASIC to structured BASIC to Visual Basic shows that the Beginner's All-purpose Symbolic Instruction Code has remained an important technology for many in personal computing. Although this user base has sometimes been neglected or denigrated as hobbyist or amateur, the technical community is an important component of the abstraction that I describe as Code Nation in this book. BASIC programming was

an essential haven for the learn-to-program movement as it weathered the multiple crises facing the computer literacy movement in the mid-1980s. These mostly self-taught gamers, hobbyists, students, and power users acquired their coding acumen by sharing resources in computing centers and reading books or magazines that helped them to develop new skills. They shared code with fellow programmers, assisted each other when possible, and persisted through incompatibilities, upgrades, and the commercial pressures brought by rising software companies in the U.S. As the early PC platforms evolved, so did the BASIC language, changing to address the critiques of industry pundits and the fast-moving cycles of product development. Although I have highlighted Microsoft Press here as a publisher of technical information for BASIC and Visual Basic programmers, there were many computer book and magazine publishers that were active during this era, and we will meet several more in the coming pages. The materials and commentary from these publishers successfully diffused programming practices throughout the PC industry, helping it to grow and consolidate.

The next three chapters explore how power users, hackers, tinkerers, and gurus wrote programs, batch files, and utilities on the PC platform, strengthening America's computing infrastructure in the process. These chapters are followed by forays into Pascal, assembly language, and C/C++ programming (Chapters 9 and 10), which highlight the creative strategies used by aspiring commercial developers on the MS-DOS, Windows, and Macintosh platforms.

PART II

HOBBYIST AND HACKER CULTURES

6 Power Users, Tinkerers, and Gurus

"Creating games and utilities—sharing them, tearing them apart, and putting them back together—was how many of us first learned to program... The development of this book followed that same spirit."

Mitch Waite et al., *Macintosh Midnight Madness* (1985)

"What is a DOS guru? A DOS guru is someone who knows how to use the MODE command. DOS gurus are also those who really get to know their machine. They may do it because they want to do it, or simply because they're bored. Basically, they like to fiddle."

Dan Gookin, *DOS Secrets: An Easy Guide to Understanding the Power of MS-DOS* (1990)

As the ranks of Code Nation swelled, programmers put their new skills to work at home, in the office, and in many facets of daily life.

The chapters in Part II explore what hobbyists, hackers, and other non-professional programmers did with their newfound coding talents, including writing MS-DOS batch files, building simple PC applications and games, increasing workplace productivity, and exploring virtual worlds through telephone systems and rudimentary networks. As an extension of the learn-to-program movement, microcomputer hobbyists and hackers needed to learn new platform skills, and they were ably assisted by programming primers, magazines, and user group meetings that revealed the new technologies and promoted their use.

We examine first the "advanced users" of personal computer (PC) platforms who explored non-professional programming contexts in the U.S. and pushed the early systems to their conceptual limits in the 1980s and early 1990s. These imaginative users were often the early adopters of PCs, and they served as information hubs for newcomers who were encountering America's digital culture for the first time. We start by assessing power users, tinkerers, and gurus in Chapter 6, and continue with hacking and cyberpunk communities in Chapter 7. In both contexts, I'll examine

how some of the first personal computing technologies were adopted by tinkerers and power users, then distributed more broadly to the general public.[1] In Chapter 8, I use computer magazines, journals, and newsletters to investigate the dynamics of programming culture and the symbiotic relationship between the leaders of PC computing companies and their customers.

I will also dive a little deeper into the attributes of the first MS-DOS, Windows, and Macintosh platforms, including the programming techniques that were used to access important features. Following James Sumner and other historians of computing, I do not define PC platforms as solely defined by computer hardware or software. Instead, computer platforms are best understood as evolving standards that include a range of hardware systems, operating system components, applications, textual traditions, user behaviors, product marketing, and industry mythologies.[2] Throughout the 1980s, IBM and numerous "clone" manufacturers developed IBM PCs and compatibles in partnership with Microsoft and other manufacturers. At first, the computers on this platform ran the CP/M and MS-DOS operating systems, the latter of which IBM marketed as "PC-DOS" on its machines. After about 1990, a significant number of PCs ran both MS-DOS and Windows, featuring applications, peripherals, and settings that collectively supported the "Wintel" platform. I define this term as IBM PCs and compatibles powered by Intel microprocessors and related subsystems, including a version of the Windows operating system.

By the mid-1980s, Apple Computer also attracted thousands of followers to the Macintosh platform, which enjoyed regular use at home, in schools, and in professional contexts. A wave of Mac developers created compelling software applications, peripherals, books, and commentary. Eventually, advanced Mac systems were capable of running Unix, Windows, and other operating systems, a flexibility that complicates the tidy division between the Wintel and the Mac platforms. In Parts II and III of this book, we will encounter the creative programmers, authors, and entrepreneurs who made it their mission to understand these platforms

1. Kevin Gotkin takes a similar approach to studying new technology through tinkerers and lesser-known groups, and I am influenced by his questions. "What do we gain by tracing… a small group of amateur tinkerers? We gain an appreciation for what makes the personal computer 'personal.' We gain insight into the way that users take up technologies and adapt them for use. We gain an understanding of how technologies acquire their social meanings." Kevin Gotkin, "When computers were amateur," *IEEE Annals of the History of Computing* 36, no. 2 (2014): 4–14, here at 12.

2. Sumner defines platforms as "constellations of standards, conventions, and expectations that influence the nature and behavior of hardware, software, producers, users, and mediators." See James Sumner, "What makes a PC? Thoughts on computing platforms, standards, and compatibility," *IEEE Annals of the History of Computing* 29, no. 2 (2007): 88.

and promote their general use. We'll also see how the platforms were shaped by corporate and commercial forces, a transformation that impacted the character of the learn-to-program movement.

6.1 Computing Terminology

The terms "power user," "hacker," "tinkerer," and "guru"' are socially-constructed labels that computer users and journalists have used since the early days of personal computing to ascribe special status to members of their technical communities. These labels came into use (or were redeployed) as microcomputers gained broad distribution and became objects of fascination for the American public. Steven Levy pointed out some of the implications of this terminology 35 years ago in his best-selling book *Hackers: Heroes of the Computer Revolution* (1984). This book and its 2010 revision depict hackers in largely positive terms. Levy describes hackers as *trailblazers*—brilliant and eccentric computing "nerds" who took social and personal risks with computers, bent the rules, and pushed the computing world in important new directions.[3]

I'd like to expand on Levy's definition by adding some additional terminology, capturing a broader range of computing experiences. These terms will be useful as we consider the various subgroups at work (and play) in American computing culture.

The term "power user" came into common use in the 1980s, as certain advocates for computing developed special skills for working with or customizing existing hardware and software. PC power users didn't simply *know* the basic features of computers, operating systems, and applications—they typically purchased and *mastered* the fastest, most powerful PCs. They knew the best ways to issue commands, perform common tasks, and find advanced information—before the development of the commercial Internet. Power users were often regular readers of *Byte*, *PC Magazine*, *Macworld*, and *Communications of the ACM*—the leading consumer magazines for American PC users. They also participated in *user groups* or online *bulletin boards*, and they often argued for the superiority of one computing platform over another. For example, a power user might be heard extoling the benefits of Macs over PCs, Windows over the Mac, or Unix over Windows. Power users were essentially *advanced users* (highly motivated experts who knew *all* the commands and features in a program), though they typically were not as proficient as *system administrators* or the professional computer engineers who designed and maintained PC systems. Power users typically knew enough about programming

3. Steven Levy, *Hackers: Heroes of the Computer Revolution* (New York: Anchor Press/Doubleday, 1984; Revised edition, Sebastopol, CA: O'Reilly Media Inc., 2010), Preface.

to create and edit simple programs, write batch files, manage system memory, and customize the settings in MS-DOS configuration files such as AUTOEXEC.BAT and CONFIG.SYS. In the late 1980s and early 1990s, power users often had experience with a dialect of BASIC, which sometimes served as a gateway to new proficiencies in FORTRAN, Pascal, assembly language, or C/C++. On the Mac platform, these coding skills might extend to innovative new products such as Apple HyperCard, a hypermedia programming tool introduced with the HyperTalk language in 1987.

"Tinkerers" were computer users who creatively adapted existing computer systems by altering, extending, or repairing them. Tinkering is often associated with *modifying* technical systems and peripherals (i.e., the "hardware"), rather than inventing and constructing new systems. An exception to this was the necessity for early microcomputer users to assemble the first computers by using mail order *kits*, a tradition that continued throughout the 1980s as the owners of "PC clones" often found it useful to assemble IBM PC-compatible computers via mail order memory units, parts, and peripherals. Even if the basic computer case and components of a "clone" computer came ready-made, it might still be useful to upgrade the floppy disk drives, install new hard disks, and add expansion boards for video support, serial communications, and extra random access memory (RAM).[4] Tinkering was often necessary just to get printers, modems, pointing devices, and other computer peripherals to come to life or work properly. Computer users who managed this work and found it rewarding might also claim "advanced hobbyist" or "guru" status, but even novice computer users were forced into modifying and repairing systems from time to time, just to make them run properly. In scholarship connected to the history of technology, tinkering is often studied as part of the "diffusion and domestication" phases of technology adoption, in which members of the public encounter and try to master new products with technical components. Tinkering is also connected to the forms of power, authority, and resistance that are associated with the deployment of new technologies.[5]

Being described as a computer "guru" was the biggest complement of all in PC culture—at least in the status-conscious circles of authors, power users, and hackers. The term guru originates in Eastern spiritual traditions—a guru is essentially an enlightened master who acts as a teacher, guide, or expert in enigmatic

4. Although Apple Macintosh computers were not "cloned" and sold through original equipment manufacturers (OEMs), Mac users still needed to tinker with peripherals and memory boards, especially when the Macintosh II systems came out in the early 1990s.

5. For a discussion of scholarly approaches to tinkering, see Kathleen Franz, *Tinkering: Consumers Reinvent the Early Automobile* (Philadelphia: University of Pennsylvania Press, 2011).

domains of knowledge. Gurus (and their Western counterparts, "wizards") are not encountered very often, but if you can pin one down their esoteric knowledge is worth contemplating. In the PC software industry, gurus were essentially the hobbyist or "consumer" versions of the corporate master programmer and accorded social status for their expertise. Gurus often worked for computer corporations but held jobs that put them outside of regular reporting structures or software development processes.

Here's another thing about gurus: in the PC industry they often knew undocumented features, hidden hardware and software tricks, and the location of "Easter eggs" (hidden messages or rewards) in software. As computer book publishing and journalism gained momentum in the PC industry, gurus were often sought out as book authors, columnists, and entrepreneurs. Stereotypically, gurus were thought of as male, soft-spoken, shy, and anti-social. While it was possible for them to ride their reputations to lucrative positions, just as often they might walk away from commercial success, distrusting the spotlight. Many gurus also lived on borrowed time: their social standing rapidly diminished when the platforms that they championed fell out of favor or lost market share, an occurrence that happened regularly in the PC industry.

Only a few polymaths found it possible to switch from one popular platform to another and retain their status as an industry expert. This was a significant issue because programmers were not spread out evenly among the various PC systems. A 1989 snapshot from the International Data Corporation (IDC) shows the following breakdown for operating system platforms in the desktop computer market place: MS-DOS 75%, Windows 14.5%, Macintosh 6.5%, Unix 2.3%, OS/2 1.7%.[6]

Power users, hackers, tinkerers, and gurus all had some knowledge of programming, but they tended to use their expertise to customize or "supercharge" existing systems, rather than create entirely new software. As such, advanced users traversed the boundaries between hobbyist and professional developer. This makes them one of the more interesting "influencer" groups to assess as we explore the expanding networks of PC programming culture. To investigate these groups and the platforms that they helped to develop, I'll discuss the writings of Van Wolverton, Dan Gookin, Andy Rathbone, Cary Lu, Mitchell Waite, and many of the publishers that worked with them. In Chapter 8, I'll return to an analysis of PC platforms by evaluating the commentary of users who wrote to popular computer magazines with their problems, including *Byte*, *Communications of the ACM, Dr. Dobb's Journal*, and *Macworld*.

6. Cited in Gary Andrew Poole, "1991 UNIX Forecast: World issues will govern its continued growth," *UNIX World*, January 1991, 70–77, here at 74.

6.2 Tinkering with Personal Computers

In June 1991, Microsoft released the MS-DOS 5.0 operating system, one of the strongest-selling versions of its original systems software for IBM PCs and compatibles. The retail price for a boxed copy of MS-DOS including software, disks, a usage license, and documentation was $99.95. (See Figure 6.1.) License packs were available for $79.95 per machine without documentation, and deeper discounts were available for corporate customers who bought in bulk. The operating system also included the QBasic Interpreter, a replacement for GW-BASIC that supported structured programming, integrated debugging, and many of the popular features of the Microsoft QuickBASIC compiler. (See Chapter 5 for a technical description of this product.) Jeff Prosise, a contributing editor to *PC Magazine*, wrote about the newest release in the most enthusiastic terms: "Without a doubt, MS-DOS 5.0 is the best MS-DOS ever... With its many time- and memory-saving features, DOS 5.0 is worth whatever you pay for it."[7] MS-DOS was nearing the peak of its market saturation and relevance for customers in the PC industry. The mainstream computing press continued to swoon over the product for almost a year, praising its potential to enhance productivity and transform PC-based computing for business and home users.

MS-DOS 5.0 eventually achieved an installed base of some 50 million users, enriching Microsoft and its global partners. A focus of branding and journalistic prose about this platform was the power and stability that it provided. By the early 1990s, the hardware for IBM PCs and compatible machines had made major strides, far surpassing the bare-bones systems of the late 1970s and early 1980s. A mid-range IBM PC or compatible in late 1991 likely included an Intel 486 microprocessor, 4MB of RAM, a 200MB hard drive, one 3.5" diskette drive, and an optional CD-ROM drive—vastly increasing the memory capacity of a typical system. (IBM's newest offering in this category was the IBM PS/2 Model 90, which featured microchannel bus architecture and a 20MHz Intel 486 microprocessor.) To round out the list of peripherals, IBM PCs and compatibles were typically supplied with a super video graphics array (SVGA) color monitor, enhanced keyboard, and a mouse or trackball pointing device. Most office work groups (and many home users) also had access to a high-quality dot matrix printer or a laser printer. (In the latter case, the industry standard was the Hewlett Packard [HP] LaserJet 4, a laser printer that sold for $2,199 in 1992 and was known for its crisp images and durability.) In short, the era of powerful mass-market PCs and peripherals had arrived, and users were buying these systems to operate small businesses, manage their taxes, play computer games, compose

7. Jeff Prosise, "DOS 5: What's in it for You?" *PC Magazine*, September 24, 1991, 223–243, here at 243. For a photo of Prosise, see Figure 10.5.

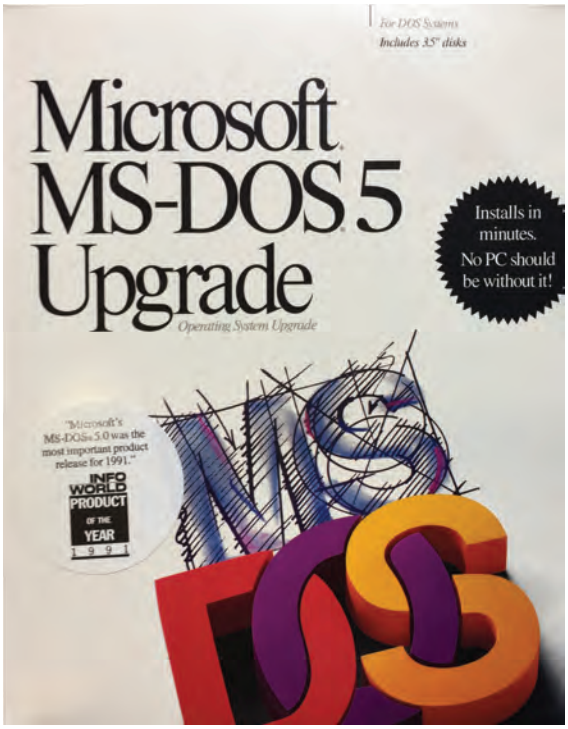


Figure 6.1 **The MS-DOS 5 Upgrade software package (1991).** (Used with permission from Microsoft)

music, edit videos, complete school work, create desktop-publishing projects, and much more.

Throughout 1991 and 1992, powerful, low-cost IBM PC-compatible "clone" computers also flooded the market from manufacturers such as Compaq, Dell, Gateway, HP, Micron, Austin, ZEOS, and numerous mail order companies. As economic historians have pointed out, the intense competition among clone manufacturers lowered costs for consumers and also increased product reliability, creating a successful new industry.[8] Virtually all of the IBM PCs and compatibles came with MS-DOS 5.0 installed, and OEMs paid Microsoft directly to license and install custom versions of the operating system software. Many OEMs also included Microsoft Windows 3.0 with their systems, the graphical operating environment released in May 1990. Windows 3.0 ran on top of MS-DOS and provided a shell through which

8. For the economic impact of clones and the overall growth of the personal computing sector by 1991, see Richard N. Lunglois, "External economies and economic progress: The case of the microcomputer industry," *The Business History Review* 66, no. 1 (Spring 1992): 1–50, here at 30.

users could perform common tasks and run built-in utilities such as Write, Paintbrush, Terminal, and Calendar. Windows 3.0 was the first commercially successful version of Windows, enhanced to use the virtual memory provided by Intel 386 and 486 microprocessors. In 1992, Windows 3.1 was released, with the TrueType font system, support for multimedia, and additional enhancements. Extending the PC/MS-DOS platform to include Windows was controversial at first, but by the mid-1990s most users had accepted the familiar graphical user interface (GUI).

MS-DOS 5.0 and Windows 3.x made PCs easier to use, but there were still problems and pitfalls. As Joseph Corn points out in *User Unfriendly*, even with a GUI the users of PCs were regularly met with steep learning curves and frustration when they experimented with new systems.[9] This was most obviously the case with users who were less experienced with computing technology, and were forced to purchase, install, and use new products on their own. Often, they had no computer experience to draw on at all, and it was bewildering for them to learn new skills or find out how to get help. While it may have been somewhat easier for newcomers to operate Macintosh computers (and Apple widely advertised this fact), Macintosh products also required considerable expertise, especially when hobbyists attempted to develop their own software for the Mac's GUI environment. (See Section 6.7 for an example of the challenges they encountered.)

As the MS-DOS, Windows, and Macintosh platforms matured, it was often computer book authors and magazine columnists who users turned to when they encountered problems. The following sections profile several of these author/programmer/entrepreneurs and the computing skills that they taught to both hobbyists and professional workers. Chapter 8 covers essentially the same time period, but from a user's perspective.

6.3 Van Wolverton and Batch Files

The author Van Wolverton (1939–) made his start with computers in the 1960s, working at IBM and Intel as a technical writer and editor. Wolverton had both rich and diverse experiences in the publishing industry, working over the years as a newspaper reporter, editorial writer, political columnist, and technical writer. (See Figure 6.2.) In the PC marketplace, Wolverton also had excellent timing. In mid-1983, when Microsoft was in the planning stages of forming Microsoft Press to publish books about its products, Wolverton sat down with employees Andrea Lewis and Nahum Stiskin to outline a "behind the scenes" guide to MS-DOS, the

9. Joseph Corn, *User Unfriendly: Consumer Struggles with Personal Technologies, from Clocks and Sewing Machines to Cars and Computers* (Baltimore, MD: The Johns Hopkins University Press, 2011).

Figure 6.2 **Van Wolverton at Intel Corporation in Santa Clara, California, where he worked as the manager of a technical writing group. This image was taken in 1979, about 4 years before Wolverton started work on the first edition of *Running MS-DOS*.** (Photo courtesy of Van Wolverton)

company's newest operating system.[10] Once Microsoft Press had been formally organized in November 1983, Wolverton started writing. His work eventually led to the publication of two books for beginning and advanced users, *Running MS-DOS* (1984) and *Supercharging MS-DOS* (1986).[11] Both books promoted the IBM PC and compatibles platform at a time when it was new and confusing for early adopters.

10. Andrea Lewis met with Wolverton first, followed by a conversation with Nahum Stiskin about the details of the Microsoft Press publication. For those familiar with the early history of Microsoft, Lewis was the original technical writer for the company and one of the two women pictured in the famed "Albuquerque" photograph of the original Microsoft employees. Stiskin helped to start Microsoft Press in late 1983 and first introduced Press authors to the extended PC community at the West Coast Computer Faire from March 22 to 25, 1984. I thank Van Wolverton for this recollection, which I received via email correspondence in July 2019. For a photo of Stiskin and early notes about Microsoft Press, see Denise Caruso, "People," *InfoWorld*, April 23, 1984, 21.

11. Van Wolverton, *Running MS-DOS* (Bellevue, WA: Microsoft Press, 1984); Van Wolverton, *Supercharging MS-DOS* (Bellevue, WA: Microsoft Press, 1986).

Although computer books about DOS were generally successful, *Running MS-DOS* became a true market leader, selling millions of copies and insuring that Microsoft Press would be profitable and could expand their operation in new directions. To maintain a leadership position in the DOS book market, Microsoft Press carefully planned and published new editions of *Running MS-DOS* for every new release of the software through version 6.22. Wolverton deserves credit as a pioneering technical writer who revised these books and popularized the commands and procedures used on DOS-based PCs. One area of interest was *batch file programming*, or coding with a rudimentary scripting protocol that allowed users to customize their systems by automating basic commands and procedures. To write batch files, DOS users needed a thorough knowledge of DOS commands, as well as some exposure to programming concepts and computational logic—roughly the amount that they would receive if they took an introductory course (or read a book) on FORTRAN, BASIC, Logo, or Pascal programming.

From earlier work at Microsoft, Wolverton had some inside knowledge about the internal features of DOS and the subtleties that users might experience when using the system.[12] Moreover, Wolverton was well supported by the staff of Microsoft Press, including JoAnne Woodcock, a master editor who worked diligently with Wolverton during each stage of the publishing process. *Running MS-DOS* was lavishly illustrated and printed in multiple colors, but the *prose* was especially sparkling—a synthesis of Wolverton, Woodcock, and, according to Wolverton, his wife Jeanne's thoughtful contributions. I observed some of this work first hand when I became the technical editor for the third edition of *Running MS-DOS* in 1988. (See Figure 6.3 for an image of the book's cover.) As a technical editor, it was my job to carefully test and verify the instructions in the book, experimenting with batch file programming on numerous systems and switching back and forth between different versions of DOS. We all knew that previous editions of the book had sold millions of copies, and it was important to make every detail as technically accurate as possible, including the command reference at the end of the book. Throughout Microsoft, we knew that many users of MS-DOS were finding the product challenging to use, and so we created a narrative and curriculum that moved from one topic to the next, allowing those with some knowledge of operating systems to learn more.

What topics were important for early MS-DOS users to know? *Running MS-DOS* taught readers how to work with files and diskettes, create and manage directories, create text files with Edlin, launch applications, and construct batch files. Batch file programming was intentionally set aside as a "power user" topic. In that section,

12. Van Wolverton, *Running MS-DOS, 20th Anniversary Edition* (Redmond, WA: Microsoft Press, 2003), xvii.

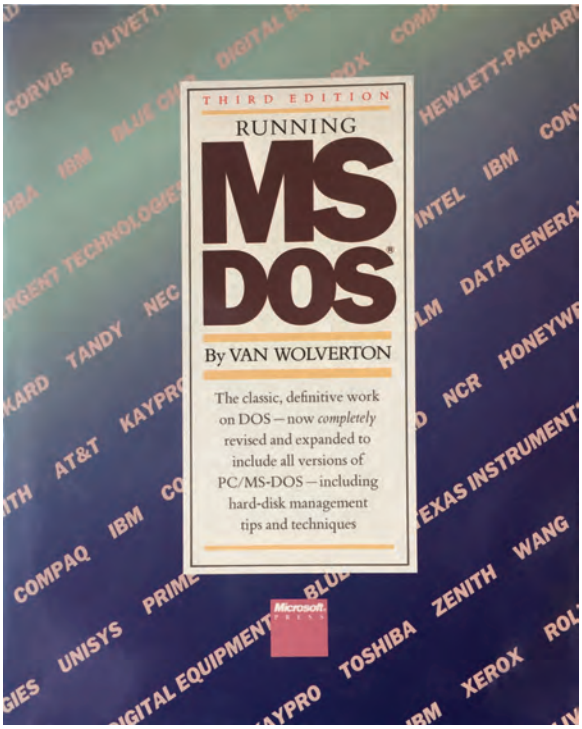


Figure 6.3 ***Running MS-DOS*, Third Edition (1988), by Van Wolverton. Updated for P-DOS/MS-DOS version 4.0, this bestselling volume helped novice and experienced users manage their systems and issue DOS commands.** (Used with permission from Microsoft)

Wolverton explained that experienced users could string together dozens of MS-DOS commands in a way that would automate common processes. When the commands were associated with a name, using the name later would essentially *extend* the MS-DOS command structure. Wolverton then explained how to manage two special settings files called AUTOEXEC.BAT and CONFIG.SYS files. These start-up files configured MS-DOS after the system "booted," and they established core system settings, which could be unique on each system. While batch file programming is technically more like "scripting" than building software with a general-purpose programming language, the process seemed non-trivial for most MS-DOS users. As a publishing category, batch file programming fit into what authors and publishers were describing as the "programming skills" classification in the 1980s. This was also the domain of the power user and advanced hobbyist. In sociological terms, we might consider batch file programmers to be the rank and file developers of Code Nation. Many moved back and forth between batch file programming, coding

in BASIC, and writing macros for popular applications such as Microsoft Excel or Microsoft Word. On a Unix/Xenix system, they would be among the people using the AWK reporting tool, which is commonly used for pattern searching and text processing.

But what was it like to enter simply MS-DOS commands? For those who haven't experienced the excitement of typing system directives at a text-based prompt, imagine that you are typing cryptic words and symbols into a text box or instant message window. Each time that you press the Enter key, the MS-DOS command that you enter is processed by the operating system. In most cases, you'll see a visual representation of the command that you entered and the work that takes place when you enter it. Once you get the hang of entering individual MS-DOS commands, you might want to try a few that support extra abbreviations and symbols to help them do their work. These modifying characters are called "parameters" in the DOS documentation. For example, to display a list of the electronic files in a folder on a diskette that is inserted into disk drive "A" in your computer, you would type the following "dir" command at the MS-DOS prompt:

```
dir a: /p
```

The trailing parameter /p is an option that forces MS-DOS to pause the directory listing after one screen of information has been displayed. This is useful if the command has produced a lot of output, because the MS-DOS output window has no visible scroll bars. Directory ("dir") directs MS-DOS to display the names and attributes of files so that you can review them and consider doing something with them in a subsequent command. (If you're wondering how to continue viewing files in an output window that is paused, you simply press the spacebar on the keyboard.)

I offer this detailed walkthrough as a way of explaining why books like Wolverton's were so helpful, and, ultimately, successful. DOS commands are essentially non-intuitive, and most PC users learned just enough to get by. Those who knew all the parameters and switches were referred to as *power users*. They had higher status and they were in greater demand than casual computer users. Not to teach simple commands like Directory, but to help perform more sophisticated tasks, like managing data on systems, backing up files, and running applications in creative ways.

Here's another example from the third edition of *Running MS-DOS*.[13] It refers to the Mode command, a tool that most users issued infrequently. Although the command is cryptic, it is also very powerful, packing a lot into a few short directives. This example relates to printing, and it also digresses into the murky world of a computer's external hardware interfaces or *ports*. To change the parallel printer

13. Van Wolverton, *Running MS-DOS*, Third Edition (Redmond, WA: Microsoft Press, 1988), 436.

attached to the first parallel printer port on an IBM PC or compatible system so that it is configured to print 132 characters per line and 8 lines per inch, you would type the following command at the MS-DOS prompt:

```
mode lpt1: 132,8
```

Why would a person use such a command? For DOS users who owned a parallel printer, the Mode command was used in this way to configure a printer for printing.[14] Such a step was often necessary before using a printer, because DOS-based computers didn't automatically "know" what types of devices were attached to them through the built-in parallel and serial ports. (This "plug and play" functionality would later be added to Microsoft Windows, but not MS-DOS.) It was easy enough to type in this command, but how did you learn the various parameters and numbers to use? It took research, often via a printer manual that might clarify the available options. Wolverton's *Running MS-DOS* was one of the best resources that users could find to do this sort of work.

DOS was not the only operating system that presented a "command-line" interface, of course. Many contemporary operating systems offered similar commands and parameters. These systems included DEC OpenVMS, Apple DOS, CP/M, and most variations of Unix. Despite the learning curve, once you learned the syntax, you could do a lot of work quickly and efficiently with these systems, because they didn't require lots of overhead to display and run a GUI. Indeed, some computer users preferred command-line systems so much that they strongly resisted the onslaught of GUI platforms that arrived in the mid-1980s, including Mac OS, Microsoft Windows, OS/2, and the X Window System (a popular GUI framework for Unix-like systems).

Van Wolverton gained great credibility as a "DOS guru" in the early years of MS-DOS, and he was esteemed by novice users and software developers alike. As the user community grew and became more experienced, publishers like Que, Sams, Microsoft Press, and International Data Group (IDG) followed with many books for power users and corporate information workers. Wolverton's *Supercharging MS-DOS* is one prominent example of this phenomenon.[15] *Supercharging MS-DOS* picked up where *Running MS-DOS* left off, providing more information about customizing

14. "Parallel" refers to the way the data was sent; IBM's parallel ports and printers typically handled 8 bits of data at once.

15. Cited already, the first edition was published in 1986 and the second edition (updated for MS-DOS 4.0) was published in 1989. In 1991, the third edition arrived, updated for MS-DOS 5.0 and featuring the work of Dan Gookin (see Section 6.4), an experienced PC author. I draw quotations from the co-authored book; see Van Wolverton and Dan Gookin, *Supercharging MS-DOS*, Third Edition (Redmond, WA: Microsoft Press, 1991).

IBM PCs and compatibles. The authors discussed using IBM's extended character set, experimenting with hexadecimal arithmetic, controlling the keyboard with ANSI.SYS, optimizing printers, using advanced graphics adapters, and managing system memory. Power users, tinkerers, hackers, and would-be gurus readily purchased this book and worked to implement Wolverton and Gookin's timely advice. Training resources of this type were not meant to be read from cover to cover, of course, but consulted when advanced users wanted to tinker with their systems, fix unexpected problems, and make the computer and its software run more efficiently. In addition, power user books helped users make their DOS-based systems appear visually different from other users, prompting admiring glances and questions like "How did you get your PC to display those colors?" or "Can you show me how to configure my system like that at startup?"

A core group of power users seemed to take shape around the batch-file writing activity, with many DOS users enjoying it so much that they eschewed taking deeper dives into programming with languages like BASIC, Pascal, or C. Internally, a batch file is simply an ASCII text file containing one or more DOS commands that the user wants to execute as a set, typically to perform some unified function. According to *The MS-DOS Encyclopedia*, batch files offer a handy mechanism for performing frequently-used commands without having to type them each time that they are needed.[16] These scripts were invoked just like regular DOS commands—users typed the name of the batch file at the DOS prompt and pressed Enter. The contents of the batch file were then executed in sequence by a special batch-file interpreter built into the command processor (known as COMMAND.COM).

The "programmable" aspect of DOS batch files is related to the option that users have to include replaceable parameters inside the batch file. These parameters hold spaces for filenames and other useful information that the user might supply on the command line when the batch file is invoked. Special batch file commands such as If, Goto, For, Pause, Echo, and Rem could be used to add programmable features to the rudimentary scripting language. These commands allowed the user to build simple decision structures, manage program flow, create textual output, and add documentation attributes—in short, the elements of a simple programming language that could control what happened within the MS-DOS operating system.

In *Supercharging MS-DOS*, Van Wolverton and Dan Gookin provided readers with dozens of clever batch files, including the Findfile.bat utility, Findfile.bat was used to locate an individual file or a pattern associated with files (such as all files with the extension .EXE) somewhere on the user's hard disk. Why would this be necessary? Although DOS users initially stored files on floppy disks, hard disks were introduced

16. Ray Duncan, ed., *The MS-DOS Encyclopedia* (Redmond, WA: Microsoft Press, 1988), 752.

with the IBM PC XT in 1983, and this storage media became a standard feature of personal computing in the following years. Hard disks, increasing in size each year, became a distinguishing feature of the PC Revolution as it quickly distanced itself from the mail-order kits and floppy-drive systems of the 1970s. Findfile.bat shows how this transition impacted the users of computers with hard disks. Importantly, it presented a capability that the MS-DOS operating system did not offer until later versions. Such a tool would come to be called a "utility" program, i.e., a batch file or application that could help users save time and increase their "productivity" (the latter term becoming an acute obsession of the period). For more about creative responses to this issue, see Section 9.2: "Inside the IBM PC with Peter Norton."

6.4 The *DOS for Dummies* Phenomenon

DOS "guru" Dan Gookin began writing computer books in 1987 with a modest title co-written by Andy Townsend entitled *Hard Disk Management with MS-DOS and PC-DOS.*[17] Typical of early books about personal computing, this volume focused on a topic that had become vexing for many: file management and organizational tasks. In the era of hard disks, a DOS-based computer inevitably had numerous application programs and data files that needed to be systematized, backed up, and cared for. Power users, professional programmers, office workers, and gamers all had to think about these tasks—few were able to use computers without organizing and safeguarding their files.

When Gookin worked on the 1991 revision to *Supercharging MS-DOS* (which Wolverton was unavailable to complete), he had already written 15 books for a selection of reputable computer book publishers. Over the next few years, Gookin would become one of the most successful technical writers of his era. After *Supercharging*, Dan wrote *Managing Memory with DOS 5* and *MS-DOS to the Max* for Microsoft Press, feeding the power user audience with insider information about how the MS-DOS 5.0 and MS-DOS 6.0 operating systems worked.[18] Gookin then went to work on a phenomenally successful group of titles for IDG Books. (See Figure 6.4 for a photo of Dan Gookin and several of these books.)

I was the Microsoft Press acquisitions editor responsible for the publication of Gookin's Microsoft books, including *Supercharging MS-DOS*, *Managing Memory*, and *MS-DOS to the Max*. I met Dan in 1990 after reading his short book *DOS Secrets*,

17. Dan Gookin and Andy Townsend, *Hard Disk Management with MS-DOS and PC-DOS* (Blue Ridge Summit, PA: TAB Books, 1987).

18. Dan Gookin, *Managing Memory with DOS 5* (Redmond, WA: Microsoft Press, 1991); Dan Gookin, *MS-DOS to the Max: Tools and Techniques that Will Make Your Hard Disk Scream!* (Redmond, WA: Microsoft Press, 1993).

Figure 6.4 **Dan Gookin in his writing studio and workshop (2014). On the shelf behind Gookin are several of his "Dummies" books.** (Courtesy of Dan Gookin)

published by Computer Publishing Enterprises.[19] Mr. Gookin is one of the funniest and hardest working authors that I have ever worked with, and I have nothing but admiration for what he was able to accomplish in the wild and woolly days of early DOS publishing. Typically, Gookin would write a 300-page book in a matter of 8 to 10 weeks, which I witnessed and supported on several occasions. He also had an incredible knack for developing technical content on timely topics, especially when they addressed real-world problems that computer users were struggling with.

For example, Gookin's *Managing Memory with DOS 5* book helped users to find extra space in RAM for their DOS-based programs. The book was so successful that an incoming MS-DOS 6.0 program manager at Microsoft vowed (in private) to modify the operating system software so that it was no longer necessary to "buy a book

19. Dan Gookin, *DOS* Secrets: *An Easy Guide to Understanding the Power of MS-DOS* (San Diego, CA: Computer Publishing Enterprises, 1990).

from Gookin" every time that a user wanted to configure memory in the best way possible. In short, Dan had figured out how do things that the current version of MS-DOS didn't do well, and he motivated Microsoft to improve their software and user experience.

Gookin acquired his information through careful investigative work and occasional contact with employees from the Microsoft product groups. Microsoft was eager to support the flow of accurate information, especially if it helped people to use the software. Book authors like Gookin basically collaborated with publishers and software teams to distribute information that might help computer corporations increase their market share. They also hoped that the books would enhance customer satisfaction and productivity with the products. The ongoing collaboration took the form of new books, magazine articles, radio show appearances, and participation at industry events such as COMDEX and Macworld Expo. (For more about industry trade shows, see Chapter 11.)

Dan Gookin's commercial success skyrocketed as the installed base of MS-DOS users grew. After a few Microsoft Press titles, he co-developed a new concept for IDG Books that became the popular *DOS for Dummies* series. Debuting in November 1991, the *Dummies* books were casual and humorous "how to" guides for beginning and new-to-topic PC users. In addition to funny and quick-witted descriptions, the first *Dummies* books featured clever cartoons by Rich Tennant. *DOS for Dummies* soon became a commercial success, with new editions published for each pending version of MS-DOS. By 1993, IDG Books reported that the *Dummies* series had over two million copies in print, a statistic they happily printed on the front cover of their books.[20] (See Figure 6.5.) Over time, the *DOS for Dummies* books appeared to catch up and surpass what even Van Wolverton's *Running MS-DOS* series had sold, although an exact sales and revenue comparison is not possible. However, IDG fully committed to the concept and hundreds of new *Dummies* books about computers, software, and popular culture soon appeared. Although they were written by other authors, most traded on the same humorous and informal style that Gookin had pioneered. At IDG, CEO John Kilcullen masterfully expanded the line of low-cost books into a global publishing phenomenon that caught the attention of all the trade publishers and made *Dummies* books into an American commodity.[21] A movement that began by selling books to DOS power users ended up transforming the U.S. publishing industry. An outstretched arm and placard on

20. Dan Gookin, *DOS for Dummies*, Second Edition (San Mateo, CA: IDG Books Worldwide, 1993). The second printing of the second edition indicated that over 2,003,267 copies of Dummies books were in print.

21. Rachel Donadio, "Dumbing up," *New York Times Sunday Book Review*, Sept. 24, 2006.

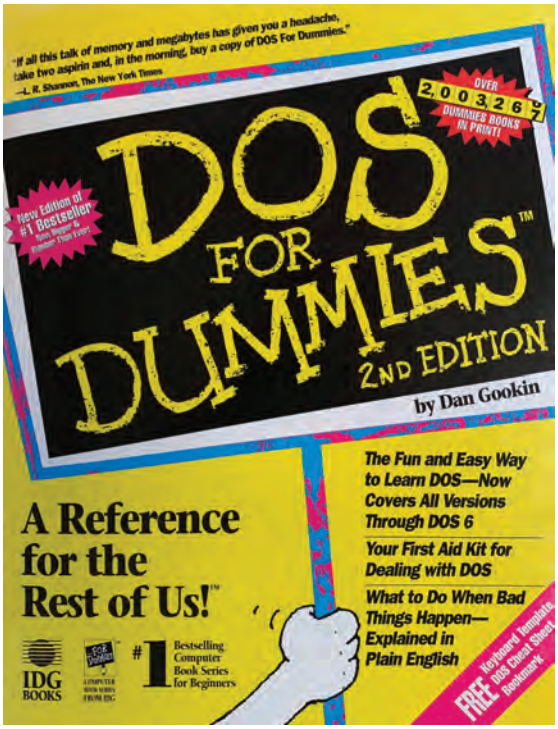


Figure 6.5 **Cover of *DOS for Dummies*, Second Edition (1993), by Dan Gookin. An outstretched arm holding what appears to be a protest sign emphasizes the idea that the book is connected to a popular social movement.** (Cover image courtesy of Wiley Publishing and used with permission)

the cover of most books even made the titles appear to be part of a popular social movement.

IDG's sales figures were certainly astonishing to those of us who witnessed them first-hand, and it must be emphasized that in the early 1990s nothing seemed "inevitable" about the runaway success of the *Dummies* series. In terms of MS-DOS, there were many good books about PC-based operating systems, and not all of them sold well, despite skilled authors, thoughtful content, quality controls, and well-executed marketing campaigns. The *Dummies* series was essentially a triumph in timing and creative coordination; several editorial and marketing factors intertwined to help the *Dummies* books attract an audience and gradually expand their market share. These factors included Gookin's timely and humorous prose (often imitated by competitors, but hard to replicate), the use of clever art and illustrations, Kilcullen's marketing innovations, the book's vibrant yellow color and branding, and a relatively low price, which came in much lower than the competition.

Moreover, there is the curious detail that many buyers seemingly had *no problem with being identified as a dummy,* a branding attribute that attracted many buyers but offended others.[22]

The most popular book in the *Dummies* series eventually became Andy Rathbone's *Windows for Dummies*, which has sold a reported 15 million copies to date, making it the bestselling computer book of all time.[23] Over the years, Rathbone has expanded his writing portfolio to include dozens of books about Microsoft's operating systems. As of this writing, the total number of *Dummies* books is approaching 200 million copies in approximately 2,500 unique titles. IDG Books changed its name to Hungry Minds Inc. in 2000, and in 2001 the *Dummies* series was acquired by John Wiley & Sons.

6.5 The Economic Impact of Personal Computers

What happened with PC software sales and book publishing in the early 1990s felt a bit like a *gold rush* in the computing industry, although in absolute terms the volume of software sales for PCs was still dwarfed by the biggest corporate-software firms in America.[24] Microsoft's revenues grew in these years from $804 million (in 1989) to $4.7 billion (in 1994). The company's fulltime employee totals surged from 4,037 to 15,017 during the same period.[25] As we have been charting through software and books, the cultural and economic impact of PCs felt tangible to Americans in these years, as did the influence of surging companies like IBM, Apple, Compaq, Dell, Gateway, Lotus Development, Word Perfect, Novel, Adobe, and Autodesk. What happened with the *Dummies* series in book publishing also took place in PC hardware sales and software publishing.

22. In the early 1990s, anecdotal evidence suggested that more women than men self-identified as "dummies" when it came to computers and books, and this association bothered many in the publishing community. However, as Joseph Corn suggested in the book *User Unfriendly* (2011), after a while the entire PC industry apparently seemed comfortable with making users feel like dummies in relation to their products. In the coming years, several publishers tried to imitate this branding, including Alpha Books (*The Complete Idiot's Guides*).

23. Andy Rathbone, *Windows for Dummies* (San Mateo, CA: IDG Books, 1992). The current edition of Rathbone's book is *Windows 10 for Dummies*, Third Edition (Hoboken, NJ: John Wiley & Sons, 2018).

24. See useful comparative statistics between PC software sales and the global software industry in Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: The MIT Press, 2003), 232. For an analysis of how sales of the IBM PC compared with products from the other business units within IBM, see James W. Cortado, *IBM: The Rise and Fall and Reinvention of a Global Icon* (Cambridge, MA: The MIT Press, 2019), 379–418.

25. Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog*, 233.

Because Internet-based data sharing was not widely available until the mid-1990s, the best mechanisms for sharing technical information about PC products continued to be books, magazines, and newsletters. These channels were supplemented by user group meetings and computer industry trade shows, where people could meet face to face. How were software products supported when more immediate concerns came up? In the early 1990s, most U.S. software companies provided some level of phone and fax machine support for their products, as well as information through emerging dial-up networks like CompuServe and America Online (AOL). The dial-up networks provided information and many of the features that would later become popular on the World Wide Web, such as email, file transfer capability, news feeds, discussion forums, and shopping opportunities.

The service and information gap in the software industry provided the perfect opportunity for authors and entrepreneurs who wanted to support new software users through their products. In such a dynamic environment, power users and gurus enjoyed special pride of place, especially in locations where new technologies were being purchased and deployed.

6.6 Cary Lu Introduces the Macintosh

For the Apple Macintosh, the author equivalents of Wolverton and Gookin were Cary Lu and Mitchell Waite. These author-entrepreneurs encouraged the rapid development of the Mac platform in the 1980s, and they published thousands of books for users who were curious about the Mac's features and the impressive new graphical operating system.

Cary Lu (1945–1997) was a well-respected Apple insider and columnist who made a name for himself in business and technical publishing with the iconic *Apple Macintosh Book*, released in 1984 to coincide with the debut of the first Mac.[26] Lu was born in Quingdao, China and arrived in the U.S. at the age of 3, settling in central California with his family.[27] Cary Lu received an undergraduate degree in Physics from the University of California at Berkeley and a Ph.D. in Biology from the California Institute of Technology. After graduation, Lu completed research in visual perception at Bell Telephone Laboratories, and these experiences led him to a fascination with television and the medium's potential to inform and shape the public. In the coming years, Lu developed short films for *Sesame Street* and other programs for children. He also helped to develop the *Nova* series for PBS, and he

26. Cary Lu, *The Apple Macintosh Book* (Bellevue, WA: Microsoft Press, 1984). A popular second edition was published by Cary Lu and Microsoft Press in 1985. In all, four editions were published.

27. Biographical details can be found in John Markoff, "Cary Lu, 51; put love of science into TV shows, books and films," *New York Times*, Sept. 29, 1997; and Paul Andrews, "Cary Lu taught, wrote about and promoted computer use," *Seattle Times*, Sept. 25, 1997.

Figure 6.6 **Microsoft Press authors Cary Lu (left) and Grant Fjermedal examine each other's work on the Microsoft campus in 1989. Lu is holding Fjermedal's book, *The Tomorrow Makers*, and Fjermedal is holding Lu's *The Apple Macintosh Book*, Third Edition.** (Used with permission from Microsoft)

regularly contributed to the programming at CBS News and NBC. In short, Cary Lu was a curious, well-travelled author who was passionate about the news, and science and technology education. (See Figure 6.6.)

In his pioneering *Apple Macintosh Book*, Lu described his introduction to the Mac as the result of intentional collaboration between Apple and Microsoft in 1983 and 1984. "This book comes out of a conversation between Bill Gates of Microsoft and Steve Jobs of Apple," wrote Lu in his preface to his "how to" book, dated February 1984.[28] As Lu describes it, Apple was finishing the user interface for the Macintosh and Microsoft was working on a supporting version of the Excel spreadsheet, which came out first on the Mac platform. Bill Gates wanted to support the Mac's initial release, and he suggested that the newly formed Microsoft Press division publish a book on the system as they had done for MS-DOS.[29] This publishing effort would help support and advertise the new Mac system, which would drive the sales of Microsoft products.

28. Lu, *The Apple Macintosh Book*, Second Edition (Redmond, WA: Microsoft Press, 1985), xi.

29. Lu, *Apple Macintosh*, Second Edition, xi.

Nahim Stiskin of Microsoft Press signed up Cary Lu, and Lu then worked with Chris Espinosa, Martin Haeberli, Mike Murray, Mike Boich, and Guy Kawasaki at Apple to document and explore the new system. (For more about the early Macintosh team and their production marketing strategies, see Chapter 11.) Jeff Harbers at Microsoft also played an important role in the project, answering Lu's questions and introducing the writer to the Macintosh application team in Bellevue. The *Macintosh Book* would join *Running MS-DOS* as two of the first books published and distributed by Microsoft Press.

On the opening pages of the *Macintosh Book*, Cary Lu echoed the words of Steve Jobs and earlier computing pioneers such as Seymour Papert and Arthur Luehrmann. Rejecting the jargon of the techno-elite, Lu wrote that computers should work the way that people do. In his mind, this was the real way to address the computer literacy problem that had been in the news.

> Computers are supposed to help us get work done quickly, easily, and effectively. So why have they become cloaked in mystique? Because most computers are difficult to use. So difficult, in fact, that we hear about "computer literacy" as if everybody must learn a new language. Computer enthusiasts haven't helped by talking computer jargon that obscures rather than clarifies. And so the mystique has grown: To work with a computer, we must think like a computer.
> Nonsense. Computers should work the way we do.[30]

Lu proceeded to explain what new Macintosh owners needed to know about their software, hardware, and decision making on a computer. Carefully-illustrated diagrams explained how the Macintosh mouse, keyboard, disks, and printers worked. Lu covered issuing commands in the GUI, setting up networks, running business applications, troubleshooting, and achieving compatibility with IBM PCs and compatibles. Of special interest to me is the book's lengthy chapter on programming languages (significantly updated in the 1985 edition), in which Lu surveys the available programming tools for the young Mac platform. Lu introduced the following products (companies in parentheses):

- Microsoft BASIC for the Macintosh (Microsoft)
- Macintosh Pascal (Think Technologies)
- Apple Lisa Pascal for the Macintosh (Apple Computer)
- UCSD Pascal (Softech Microsystems)

30. Lu, *Apple Macintosh*, Second Edition, xv.

- Modula-2 for the Macintosh (Modula Corporation)
- Logo for the Macintosh (Microsoft/Logo Systems of Montreal)
- LISP for the Macintosh (ExperTelligence)
- MacForth (Creative Solutions)
- MasterForth (Micromotion)
- Neon (Kriya Systems)
- Mac Cobol for the Macintosh (MicroFocus)
- MacFortran (Absoft)
- PortaAPL for the Macintosh (Portable Software)
- Macintosh Assemble/Debugger (Apple Computer)

While the compilers were designed to produce programs that would run under the original Macintosh operating system (a system eventually named "Mac OS"), developers needed to consult detailed Apple system manuals if they wanted to take advantage of the unique features of the new platform. These specifications they could find in a three-volume series published by Apple Computer entitled *Inside Macintosh* (1985). Also essential was a two-volume series written by Stephen Chernicoff entitled *Macintosh Revealed* (1985).[31]

It may seem surprising that so many programming languages for the Macintosh were available within 18-months of its initial release. However, as with the IBM PC and compatible platforms, there was still little in the way of commercial software available for early Mac customers, and the argument over which programming language would be dominant was far from settled. The first software publishers for the Mac were used to selling interpreters and compilers to early adopters. After all, most PC users in 1985 realized to create anything really interesting on a microcomputer you needed to create your own software. However, Lu raised some objections to the common wisdom of his day. Instead, he asked, "Do you need to program at all?"

As noted in Chapter 4, the public position of Apple in the mid-1980s was that most people needed to learn how to use application software but they didn't need to learn how to write computer programs. Cary Lu agreed. "For most people [who wonder if they need to code], the answer is no. Increasingly sophisticated application programs will fill all common requirements, including most tasks previously accomplished by writing programs. So the majority of microcomputer users will

31. Apple Computer, Inc., *Inside Macintosh*, Volumes I, II, and III (Reading, MA; Wokingham: Addison-Wesley, 1985); Stephen Chernicoff, *Macintosh Revealed*, 2 vols. (Hasbrouck Heights, NJ: Hayden Book Co., 1985).

never write programs—at least not in the traditional sense."[32] However, Lu conceded, "If you need functions that packaged software cannot provide, or if you are simply curious, by all means learn to write your own programs."[33]

Probably for these reasons, Cary Lu never wrote a programming primer. However, his instruction for new users, power users, and office workers was much admired, and he helped the Macintosh platform gain much needed momentum in the 1980s. Erik Sandberg-Diment of the *New York Times* wrote about his signal contribution: "The best book I have seen so far is *The Apple Macintosh Book* by Cary Lu." The consummate professional went on to become a founding editor of *High Technology* magazine, a technology editor for *Inc.* magazine, and a regular contributor to *Macworld*. Lu's early death at the age of 51 was deeply felt in the Macintosh community. "Cary's interests and talents were very broad," wrote Bill Gates in a statement for Lu's obituary. "I am proud to be a part of an industry that was enriched by his contribution."[34]

6.7 The Waite Group's Macintosh Primers

Although Cary Lu believed that most Mac users would eventually be free of coding responsibilities, Mitchell Waite (1946–) wasn't so sure. Mitchell ("Mitch") Waite was also an Apple insider who had seen up close what innovative programmers could do with their systems. He was one of the original owners of the Apple I computer, which he purchased in 1976 after seeing the system at a Homebrew Computer Club meeting.[35] The early community of Apple I users was relatively close-knit, so Waite was able to interact directly with Steve Wozniak and Steve Jobs about the hardware and how he was using it.[36] These insights helped Waite understand how important microcomputers were, and he became one of the first author-entrepreneurs in the San Francisco Bay Area to write popular books about electronics and microprocessor-based systems.

Mitch Waite's formative years were all in Northern California. He studied Physical Sciences and Mathematics at the College of Marin in Kentfield, California from 1968 to 1971, followed by a curriculum in Physics at Sonoma State College from

32. Lu, *Apple Macintosh*, Second Edition, 123.

33. Lu, *Apple Macintosh*, Second Edition, 123.

34. Markoff, "Cary Lu, 51" *New York Times*, Sept. 29, 1997.

35. Waite purchased the system at The Byte Shop in San Rafael, California, an electronics store opened by Paul Terrell in late 1975. The Byte Shop was the first retail store to sell the Apple I computer. Eventually, Byte Shops opened throughout the region.

36. The Apple I story and other biographical details can be found in "Mitch Waite," interview by Leo Laporte, *Triangulation*, Episode 252, June 6, 2016. https://twit.tv/shows/triangulation/episodes/252. Accessed August 22, 2019.

1971 to 1974. During these years, Waite was fascinated with electronics, and he spent much of his spare time shopping for circuits and spare parts at U.S. Army surplus stores. After experimenting with oscilloscopes and brain wave-measuring devices, he co-authored his first book with Michael Pardee entitled *Sight, Sound, and Sensation* (1974). The paperback explored DIY electronics projects including creating art with an oscilloscope, generating electronic music, monitoring muscle-wave biofeedback, running a laser-light show, and experimenting with extra-sensory perception (ESP).[37] Waite then became a technical writer for a digital telephone systems company, preparing manuals for Codec-based digital PBX systems. In 1976 and 1977, he co-authored two more books with Michael Pardee about emerging microcomputer technology, describing early kit-type systems and the microprocessors that animated them. (See Figure 6.7.) These experiences motivated Waite to leave the telephone company and write computer books fulltime.[38]

Waite was deeply interested in microcomputer programming, and he wanted to publish books about the subject for hobbyists and other non-academic audiences. Responding to what he perceived as a lack of information about assembly language, BASIC, and early operating systems, he founded the Waite Group in 1977. The small company specialized in primers and reference works for the microcomputer stage of the learn-to-program movement. Often the books would have enigmatic titles, such as *The Soul of CP/M* (1983), an assembly language primer written by Waite and Robert Lafore that explored the hidden worlds of CP/M application development.[39] Over the years, the Waite Group employed the services of dozens of experienced teachers and technical writers, including Robert Lafore, Ira Lansin, Dan Putterman, Don Urquhart, and Chuck Blanchard.[40] Several members of this cohort taught physics, astronomy, and computing courses at Marin College, where Waite had attended school in the late 1960s. Like Robert Albrecht, Waite surrounded himself with ambitious collaborators who could work quickly and well to satisfy the region's thirst for quality programming titles. By the time the

37. Mitchell Waite and Michael Pardee, *Sight, Sound, and Sensation* (Indianapolis, IN: Howard W. Sams, 1974).

38. Mitchell Waite and Michael Pardee, *Microcomputer Primer* (Indianapolis, IN: Howard W. Sams, 1977); Mitchell Waite and Michael Pardee, *Your Own Computer* (Indianapolis, IN: Howard W. Sams, 1977). For context and sales figures, see Antic Podcast, "Interview with Mitch Waite," conducted by Kevin Savetz, June 16, 2016. https://www.youtube.com/watch?v=x1dU7b4ZkHA. Accessed August 22, 2019.

39. Mitchell Waite and Robert W. Lafore, *The Soul of CP/M: How to Use the Hidden Power of Your CP/M System* (Indianapolis, IN: Howard W. Sams, 1983).

40. The Waite Group, "Leaders in Best-Selling Computer Books," [Marketing Brochure], Greenbrae, CA, 1984.

Figure 6.7 **Photo of Michael Pardee (left) and Mitchell Waite viewing a computerized math program for children and discussing their current writing projects. An accompanying article written by Don Keown was published in the *Marin Independent Journal* on December 8, 1977.** (Photo credit: Alfred M. Arn. Used with permission of Marin Independent Journal Copyright ©2019. All rights reserved)

publishing company was acquired by Simon and Schuster in 1996, the Waite Group had evolved into Waite Group Press, and they had produced over 100 books for an expanding national audience.

Although Waite never worked at Apple as an employee, he was closely connected to Apple staff members, and he was privy to early information about the Apple II, Lisa, and Macintosh computers. When the first Mac came out in 1984, Waite was in a strategic position to introduce power users and programmers to the system, and the Waite Group published a series of innovative titles about it. The Waite Group began with Microsoft Press primers about BASIC programming, including *Microsoft Macinations: An Introduction to Microsoft BASIC for the Apple Macintosh* (1985), and *Macintosh Midnight Madness: Utilities, Games & Other Diversions in Microsoft BASIC for the Apple Macintosh* (1985). (See Figure 6.8.) About the choice of language, Waite recalled, "Personal computers were a great platform for learning to program. At that

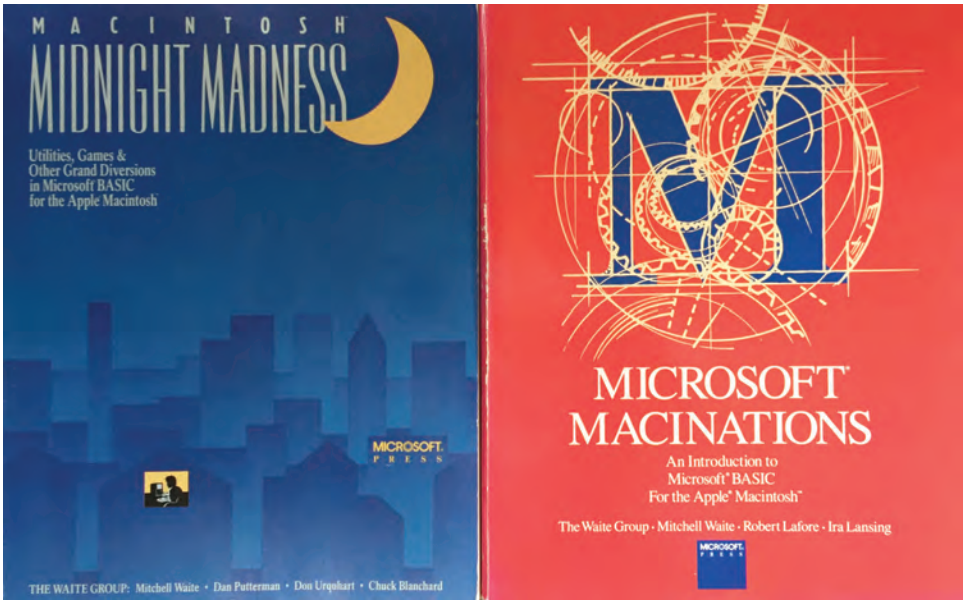


Figure 6.8 **The Waite Group's *Macintosh Midnight Madness* and *Microsoft Macinations* (both published in 1985) helped BASIC programmers to create innovative applications for the new Macintosh operating system.** (Used with permission from Microsoft)

time, the most popular language was BASIC."[41] Their books described Macintosh BASIC 2.0 for the Apple Mac, a powerful interpreted version of BASIC that sold for $160 and allowed users to write and test programs using the graphical operating environment of the Macintosh. I analyze some of their work here to show what the development process was like for Mac enthusiasts in the mid-1980s. The Waite Group's tutorials present some of the earliest examples of programming on a PC with a GUI.

What was early Mac programming like in BASIC? According to industry analysts, Microsoft BASIC 2.0 provided direct access to the system's intriguing features such as mouse support, pull-down menus, windows with sizing bars, dialog boxes, buttons, and field editing.[42] The product also gave new programmers a chance to experiment with event-driven programming concepts and enhancements related to graphics, sound, animation, and other gaming features. Microsoft BASIC

41. Antic Podcast, "Interview with Mitch Waite," conducted by Kevin Savetz, June 16, 2016. https://www.youtube.com/watch?v=x1dU7b4ZkHA. Accessed August 22, 2019.

42. Glenn A. Hart, "Microsoft Basic 2.0 for the Mac," *Creative Computing* 11, no. 5 (1985): 46–52, here at 51.

2.0 also introduced user-defined subprograms, which allowed for local variables, parameter passing, and re-usable libraries of common routines. These were features associated with structured programming, an important transition that I discussed in Chapter 5. If programmers were not ready for these concepts, however, they could continue to use the GOTO, GOSUB, and RETURN statements to control the branching of their programs.

The Waite Group plunged readers directly into this evolving world of programming conventions and GUI features. Like their earlier books, *Microsoft Macinations* was lavishly illustrated, with numerous diagrams explaining operating system concepts and language features. The book was organized into two parts. The first aimed at teaching the fundamentals of the BASIC language—the variables, graphics, loops, decision structures, subroutines, strings, and arrays that hobbyists would need to know about as they created simple games and utilities. The second part explored the new features of Microsoft BASIC 2.0, including the menus, windows, sound, and graphics capabilities discussed above. The authors also exposed readers to QuickDraw, the Macintosh's built-in library of graphics routines. In this context, they taught programmers how to control mouse clicks, graphics animation, multi-channel sound, and disk file access. The exercises were designed for the original "classic" Macintosh system, powered by the Motorola 68000 microprocessor with 128K and 512K memory configurations.[43]

The Waite Group carefully oriented programmers to the GUI of the Macintosh. They explored pixels, the coordinate system, the PSET (pixel set) command, and the LINE (draw line) statement. Their prose was friendly and colloquial, inviting readers to appreciate the riches that the lay within the new systems:

> If you view the Macintosh desktop up close you will notice that certain diagonal lines seem to be made up of little dots. These little blips are the secret of how the Macintosh's designers made their fortune. Simply said, every object, every letter, every number, everything or anything displayed on the Mac's screen is made up of dots. Because of the way the Mac was designed, you can't see these dots when they are side by side in perfect vertical or horizontal lines, but lines on an angle show them clearly.[44]

After a discussion of the coordinate system, which arranged pixels into columns and rows on the screen, the authors put these principles to work in various programming examples.

43. The Waite Group, Mitchell Waite, Robert Lafore, and Ira Lansing, *Microsoft Macinations: An Introduction to Microsoft BASIC for the Apple Macintosh* (Bellevue, WA: Microsoft Press, 1985), xii.

44. Waite Group, *Microsoft Macinations*, 72.

The following routine from *Macinations* used the LINE, WHILE... WEND, MOUSE, IF... THEN, and PSET keywords to draw pictures in the Mac's Output window with simple mouse drag motions. The code draws a line in the window from the pixel in column 400, row 10 to the pixel in column 400, row 280. When the user drags the mouse to the left of this vertical line, their output appears in the form one might see in a painting program like MacPaint. When the user clicks to the right of the vertical line, the program terminates. Here is the program code:

```
LINE (400, 10) - (400, 280)
WHILE x < 400
    x = MOUSE(1)
    y = MOUSE(2)
    IF MOUSE(0) <> 0 THEN PSET(x,y)
WEND
```

In this routine, WHILE... WEND is a looping structure, included as a way to make the program repetitive.[45] The WHILE structure directs the program to continuously monitor mouse activity and then draw something when the user holds down the mouse button. WHILE begins the loop and WEND ends the loop. The process continues until the user clicks to the right of the vertical line that has been drawn in column 400 of the Output window. The MOUSE function is used with three different arguments (i.e., the number in parentheses). If the mouse button is down, the PSET command is used to darken a pixel at the current location of the mouse pointer. The x,y coordinate for the pixel is stored in two integer variables named x and y, which hold the last location the mouse was at when the MOUSE(0) function was executed. For each example there is a program listing and one or more sample screen shots.

The second Waite Group book, *Macintosh Midnight Madness*, attempted to build on the momentum created by Microsoft BASIC 2.0 in the marketplace. Although the primer also welcomed new programmers to the development community, it was really designed to be a "construction set," allowing more experienced developers to go a bit farther and take full advantage of the feature-rich, event-driven programming language. The book's reading line, "Utilities, Games & Other Grand Diversions in Microsoft BASIC," was printed below the title on a cover that showed a darkened city scape with a sole computer user programming a Mac in what appeared to be the wee hours of the morning. The gender of the programmer is unspecified, but within the book most, if not all, of the gaming examples refer to male stereotypes, such as wizard, soldier, paratrooper, or simply a "gentleman."[46]

45. Waite Group, *Microsoft Macinations*, 145.

46. The Waite Group, Dan Putterman, Don Urquhart, Chuck Blanchard, *Macintosh Midnight Madness: Utilities, Games & Other Grand Diversions in Microsoft BASIC for the Apple Macintosh* (Bellevue, WA: Microsoft Press, 1985), 191.

In the course of 425 pages, *Macintosh Midnight Madness* presented 17 complete BASIC programs and detailed descriptions. A typical program listing spanned 10 pages or more in monospace font (over 50 lines per page), and so the programs could stretch to 500 lines or more of code. Programs of this length presented problems for hobbyist coders to type in and test, especially using the era's modest debugging tools. (There were no "autocorrect" or "Intellisense" features in the product, and while the Waite Group explained what coding errors were, they didn't emphasize debugging strategies in these titles.) As a partial solution to the challenge of typing in code by hand, Microsoft Press offered a companion disk for *Midnight Madness*. However, the disk cost $19.95 plus tax and shipping, and a note on the order card indicated that the fulfilment process could take up to 4 weeks. (The retail cost of the book itself was cheaper than the disk at $18.95.) As I discussed in Chapters 4 and 5, programmers in the 1970s and early 1980s expected to type in code from program listings using their own keyboards. Indeed, they found the process much more agreeable than using punch cards, paper tape, or panel switches to enter programs as their forbearers had done. But bound-in floppy disks and CDs would soon change this system for the better, even before the commercial Internet made downloading software a relatively simple task.

As an example of the Macintosh programs that readers of *Midnight Madness* created, let's examine the MacAnimate application. MacAnimate is a graphics editing utility that allows users to create individual bitmapped images with a frame editing tool, then animate the images by displaying them in rapid succession. The program mimics some of the picture-editing functions popularized by MacPaint, an application developed by Bill Atkinson, a member of the original Macintosh design team. In particular, MacAnimate imitates the "FatBits" feature of MacPaint, which allowed users to create bitmapped images and save them in a frame gallery. FatBits is essentially a magnification mode in MacPaint where users are able to click individual pixels in a "zoomed-in" format. This way of working was adopted by most graphics editors in the personal computing world. The Waite Group mimicked it via subroutines that creatively used the PUT, PSET, LINE, PENMODE, and PAINTRECT commands in Macintosh BASIC 2.0.

The MacAnimate code listing spans 12 book pages and some 550 lines of BASIC program code. To introduce the program, the authors described the animation utility's features and overall design, then stepped readers through a three-page "program outline," which offered a structured overview of the program's subroutines.[47] The outline resembles *pseudocode*, or a high-level description of a routine's operating principles. This description was tailored to the Mac's unique

47. The Waite Group, *Macintosh Midnight Madness*, 160–162.

system requirements, resources, and memory constraints. In particular, the authors recognized that when the program is executed on an original Mac with 128K of RAM, the graphics and animation effects would likely consume all of the system's memory. They solved this problem by creating a system of memory "overlays," which could be swapped in and out of the memory as needed. This coding tactic would minimize the danger of memory overflows, and it would allow users to enter up to 18 individual bitmapped images for their animation effect.[48]

The memory problem was not unusual in the coding context of early PCs. Programming with BASIC and other languages often involved memory management issues. In this case, the problem was exacerbated by the sizable bitmap array needed to store the animation images. If readers had access to a more recent "Fat Mac" machine (a Mac equipped with 512K of RAM), then the memory issue would not be a problem.[49] Still, by respecting that some users still had 128K machines, the authors provided useful information for developers learning how to write code in more professional contexts. As we have noted elsewhere, self-taught programmers rarely had any formal training in operating systems theory or memory management techniques.

Interestingly, the authors included few truly "structured" program elements in their code, despite the program's length and sophistication. For example, although Microsoft BASIC 2.0 provided support for user-defined subprograms (including the new SUB, END SUB, and SHARED keywords), these procedural elements were not used in MacAnimate. Instead, the GOSUB and RETURN statements defined subroutines in the old way and helped to direct program flow. In addition, no local variables were defined or used. Apparently, the concept of using procedures for structured programming was so new for the Waite Group that they didn't take the time to rewrite their code to take advantage them. The team's first Mac book, published in the same year, had this to say about the "new" concept of subprograms:

> There is one other class of Microsoft BASIC program statements that allow better organization and structuring. The *subprogram statements*, consisting of SUB, END SUB, EXIT SUB, and SHARED, were added to Microsoft BASIC for the Macintosh to make our programming more powerful. We won't go into detail about these because they are quite advanced. They are especially useful for creating libraries of routines that can be merged into our BASIC programs without conflicts. For example, a good use for a subprogram would be to create an additional class of graphics statements, perhaps turtle graphics.[50]

48. The Waite Group, *Macintosh Midnight Madness*, 159.

49. The Waite Group, *Macintosh Midnight Madness*, 150, 163.

50. The Waite Group, *Microsoft Macinations*, 174.

To be truthful, the new subprogram statements (SUB, EXIT SUB) were not more difficult to understand than the old subroutine model (GOSUB, RETURN). In fact, subprograms are much clearer to read and easier to use. They conform to the procedural conventions introduced with Pascal and C programming, and they had been in general use in minicomputer environments since the 1970s. This omission provides some insight into how the practices of computer scientists and professional developers were gradually making their way into the power user and hobbyist communities. The Waite Group would soon be highly skilled at structured programming, writing modular, object-oriented programming books on C++ and Java in the years to come.

In the history of BASIC, Microsoft BASIC 2.0 for the Apple Macintosh was an innovative but ultimately transitional product, containing some of the newer elements of structured programming and access to the Mac's intriguing new interface. It was not, however, a compiler, and for this reason it was only of limited use for professional software development. This enhancement would need to wait for Microsoft QuickBASIC 1.0 for the Apple Macintosh, a true compiler product for the Macintosh community that debuted for Mac System 6 in 1988.[51] This is the product that David Rygmyr and I used in our *Learn BASIC for the Apple Macintosh Now* book, published in 1990. (For more information, see Chapter 5.)

6.8 The Maturing Mac Platform

By the early 1990s, hardware for the Macintosh platform changed dramatically, improving the computing experience for Mac users in ways similar to what power users on the "Wintel" platform gained with new machines. In mid-1991, Apple sold a range of desktop Macs, including the Macintosh IIfx, Macintosh IIci, and Macintosh SE/30. The first two systems were designed to use replaceable, "stand-alone" monitors. These configurations replaced the built-in monitor setup that was typical of the "classic" Macs. A mid-range Macintosh IIci with 5MB of RAM, an 80MB hard disk, a stand-alone color graphics monitor, enhanced keyboard, and mouse was priced at $5,969.

The new Macintosh systems became popular in businesses and schools, and many offices also purchased a mid-range Apple laser printer, such as the LaserWriter IINT, which retailed for $3,999 in mid-1991.[52] Home Mac users certainly coveted the high-end laser printers, but often settled for the Apple ImageWriter II, a dot-matrix alternative with a retail price of $595. Even with the

51. For a list of features, screen shots, and coding examples, see Microsoft Corporation, *Microsoft QuickBASIC User's Guide for Apple Macintosh Systems* (Redmond, WA: Microsoft Corporation, 1988).

52. See "MacBulletin," *Macworld*, June 1991, 17.

lower-priced peripherals, it was obvious that Macs were more expensive than similarly equipped DOS/Windows machines. The open question was whether Macs were more powerful than DOS-based PCs, and if they offered more bang for the buck. Industry pundits sometimes believed so, and there were many debates about this question in computer publications. The price wars reached their zenith in the mid-1990s, when magazines like *Macworld* and *PC Magazine* published regular, grid-based price comparisons in their product review articles. These highlighted what the editors believed to be the "best values" on each platform. *PC World* flagged their recommended products with "Best Buy" badges. *PC Magazine* displayed "Editor's Choice" award markers to highlight the products recommended by their columnists. Often the winners of "best-values" awards were also prominent advertisers in the magazines—a situation that suspicious readers often highlighted in "Letters to the editor."[53] For more information about these user complaints, see Chapter 8.

In July 1991, Apple released System 7, its newest Macintosh operating system, which offered an improved Finder, built-in file-sharing, the HyperCard 2.1 database system, and a sharing feature that allowed programs to exchange data automatically.[54] Apple hoped that this release would solidify its power user base and make inroads against the dominant Wintel platform.[55] New Mac users also received several ease-of-use features, including a new learning feature called *balloon help*, which provided contextual assistance with unfamiliar elements of the user interface. This tool displayed short messages in a popup window when the user hovered the mouse over a new or unfamiliar feature.

Like Windows-based PCs, Mac System 7 users were encouraged to *multitask* as they ran applications in the system, loading one application after another and then switching between them, sharing information via the Clipboard as needed. Some users found the multitasking feature bewildering, as they had on Windows systems. In December 1991, Ben Smith, a technical editor and writer at *Byte* magazine,

53. "Your editorial staff has jumped onto the Microsoft Windows bandwagon whole-hog! *PC/Computing* is beginning to look more and more like a Microsoft in-house newsletter than a magazine for the general computing public." Andrew Paul Tannen [Bronx, New York], "Letters," *PC/Computing*, May 1991, 25–26.

54. The System 7 upgrade cost $99 for single computers; a group upgrade kit was priced at $349. For an introduction to the features, see Jim Heid, "Getting started with System 7: A hands-on look at some of System 7's most powerful new features," *Macworld*, July 1991, 269–278.

55. Midway through 1991, industry analysts predicted that Macintosh sales for 1991 would come in at slightly less than 2 million computers in the U.S., compared to 21 million IBM PCs and compatibles during the same year. See Sandy Reed, "Apple scouts ahead with System 7," *PC/Computing*, August 1991, 40–42.

raised a fascinating objection to early multitasking features *on all PC systems*, which included those marketed by Apple, Microsoft, and Unix-based software publishers. His comments appear to be among the first to publicly raise concerns about multitasking as a socially disruptive force. Smith warned that too much multitasking would seriously erode concentration levels and produce a breakdown in personal communication, if not productivity. This worry has been echoed by more recent writers in relation to smart phone use, social media, and the dangers of so-called "ubiquitous computing." I quote Smith's *Byte* comment in full, because it gives us some insight into the computing culture of the early 1990s:

> When we are working in a multitasking windowed environment, we have concurrent paths of communications with many programs, and we begin to think in a multitasking sort of way. Rather than focusing on a single task at hand, we widen our focus to several tasks.
>
> The mental difference is somewhat similar to the difference between conversing with one other person and carrying on conversations with a number of people at the same time.
>
> Some of us find it impossible to get any content from conversations with more than just a few people. We all have some upper limit at which the relationships make it impossible to get any information from the subjects of conversation. Likewise, with computer interfaces, there is an upper limit of productive complexity, and that limit moves upward with experience.
>
> The dangerous aspect of becoming accustomed to a complex multitasking endowed computing environment is that it can become habit forming. As we learn how to accept one level of information, our appetite increases.
>
> Eventually, we will eliminate the richest aspect of our lives: the real world around us.[56]

For historians of personal computing, the point may be that the early 1990s brought a new surge of products and commercial practices to the marketplace, and these helped to spur a widening interest in computing at home and at work. Sophisticated technologies that had been developed earlier in academic, corporate, and military contexts were now placed in the hands of hobbyists, home users, office workers, and self-taught developers who had little or no experience with computing.

56. Ben Smith, *Byte*, December 1991, 372. Smith was also the co-author of a popular Unix book about Unix System V, Release 4. See Ben Smith, *UNIX Step by Step* (Carmel, IN: Hayden Books, 1990). For warnings about multitasking overload in the era of smart phones and the Internet, see Nicholas Carr, *The Shallows: What the Internet is Doing to Our Brains* (New York: W.W. Norton, 2011); and Ashesh Mukherjee, *The Internet Trap: The Costs of Living Online* (Toronto: University of Toronto Press, 2018).

The interests of newcomers diverged significantly from what earlier experts had judged important in corporate computing environments. We are still learning what these interests were. What seems clear is that power users, tinkerers, and gurus all helped to ease the transition to these new systems. Computer book authors like Van Wolverton, Dan Gookin, Any Rathbone, Cary Lu, and the Waite Group played important roles in the diffusion of new knowledge and how the complexity of new systems might be managed. One measure of their success is the number of books that they sold to supporters of MS-DOS, Windows, and Macintosh systems.

In the next chapter, we'll study the creative output of hackers, cyberpunks, and other occasional programmers who spoke from the margins of polite computing society. Initially seen as outcasts, hackers and cyberpunks contributed to the success of America's digital infrastructure in important ways, transforming inherited computing mythologies into new products and creating a new cultural synthesis. They responded to, and protested against, the dominant social, technical, and economic worldviews that were emerging in American society.

7 Hackers and Cyberpunks

"Like many computer enthusiasts and almost all hackers, I taught myself how to program—from reading books and looking at other people's programs, and by asking questions of friends. For the first six months, I kept busy learning BASIC… This kind of learning was a welcome change from schoolwork."

Bill Landreth, *Out of the Inner Circle: A Hacker's Guide to Computer Security* (1985)

"We're dealing with the cultural manifestations of the P.C. revolution."

Alison Bailey Kennedy ["Queen Mu"], *New York Times* (1990)

This chapter continues to explore the contours of America's personal computer (PC) programming culture with an assessment of a group that is sometimes held at arm's length from polite computing society—the activists and illicit users known colloquially as *hackers*, *phreakers*, *cyberpunks*, and *cypherpunks*. Throughout the 1980s and 1990s, many computer users and virtually all security specialists feared hackers—often for good reason. In fact, this iconic subgroup has existed since at least the 1960s, holding positive and negative stereotypes in the popular imagination.

In sociological terms, *hackers*, *phreakers*, *cyberpunks*, and *cypherpunks* all speak from relatively *marginal* positions in computing society. They respond to, and protest against, the dominant social, technical, and economic structures that emerged in the U.S. in the 1980s, 1990s, and 2000s. As others have argued, I will suggest that marginalized voices within a subculture are important for historians to appreciate if we want to assess the comprehensive structures in society that control and delineate power relationships.[1] Actors from the margins often define new language, cultural practices, and anxieties about society and its preoccupations. In a technically-saturated world, marginalized or illicit group members help us to

1. For a useful expression of these ideas, see Barbara A. Hanawalt and Anna Grotans, eds., *Living Dangerously on the Margins in Medieval and Early Modern Europe* (Notre Dame, IN: University of Notre Dame Press, 2007).

understand how computer users far from the centers of power reacted when one or more technologies were made dominant over another.

As early as 1984, Steven Levy recognized that the term "hacker" had become a label of derision in American society, implying unprofessional or illegal activity (electronic intrusion) via computing systems. In *Hackers*, Levy hoped to restore the label as an appellation of honor, suggesting that *true* hackers cultivated a philosophy of sharing, openness, decentralization, and social improvement. For better or worse, this alternate definition of "hacker" has not generally gained momentum in the English-speaking world, and in our modern age obsessed with computer security, identity theft, and voter fraud, "hacking" continues to have largely negative connotations.

To explore the concepts of hacking and marginalization, I'll examine the life and activities of a prominent hacker from the 1980s that Steven Levy didn't introduce—Bill Landreth (1964–), a California teenager who was able to "crack" some of the most secure computer systems in America but ended up a convicted felon. Landreth's story shows how Americans gradually came to grips with the threat of computer intrusion in the 1980s, and how compelling it could be for programmers who learned to explore primitive communication networks with a dial-up service and a modem. We'll also meet Judith Milhon or "St. Jude" (1939–2003), a self-taught programmer, journalist, and hacker in the San Francisco Bay Area who was also an advocate for free speech, equal rights, and women in computing. Milhon was arrested for civil rights activities (protesting) and largely marginalized by the mainstream media for her controversial ideas and activities. She was a professional programmer, but also had deep roots in counterculture movements, shifting from work on Berkeley Software Distribution (BSD) Unix, to neighborhood projects with Community Memory, to books and magazine articles that advocated for sexual freedom, experimentation with drugs, and an exploration of the aesthetics of hacking—all within a popular computing framework.

The histories of hackers like Landreth and Milhon are just as important to the character and expansion of Code Nation as are the narratives of more recognizable authors, inventors, and programmers, because the margins are ultimately connected to the center, and the center is connected to the margins.

7.1 Bill Landreth and 1980s Hacker Culture

Bill Landreth was a California teenager who grew up in the 1970s and purchased his first home computer in 1979, with financial assistance from his parents. The computer was an early TRS-80 Model I (Level II), which came equipped with 16K of memory and a cassette drive for data storage. (See Figure 7.1 for a similar system.) According to his recollections in the book *Out of the Inner Circle* (1985), Landreth

Figure 7.1 **The Tandy TRS-80 microcomputer with video display, keyboard, Exatron printer, and a disk device. This system was popular among early hackers and self-taught programmers, and it featured a popular adaptation of BASIC.** (Courtesy of the Computer History Museum)

spent his after-school hours learning how to operate the computer and for the first 6 months his companion was BASIC. "There was always a new command to learn that would make programming a bit easier, and as I became more familiar with BASIC, there were all sorts of tricks I could figure out to make my programs run better or faster..."[2]

After Landreth learned BASIC, he moved on to assembly language for the Z-80 microprocessor, and he eventually learned about mainframe and minicomputer operating systems, which he saw as the gateway to massive amounts of information. Landreth began asking his friends about their experiences with computers, and he eventually met people who knew about telecommunications and early bulletin boards. From a hobbyist magazine, Landreth found telephone numbers that he could use to connect to mainframe computers via a modem, and his circle of

2. Bill Landreth, *Out of the Inner Circle: A Hacker's Guide to Computer Security*, with Howard Rheingold (Bellevue, WA: Microsoft Press, 1985), 10.

friends eventually gained access to local companies through the phone lines.[3] Over time, Landreth traded information with several other hackers and they located accounts that they could enter by guessing the passwords. Once an account had been "hacked," it was often good for 6 or 7 months (until a user changed the password or the account was shut down).[4]

Landreth chose "The Cracker" as his handle and used that pseudonym on all bulletin boards to establish a reputation among other hackers. In early 1982, he and other coders formed a group called "The Inner Circle" and they tried for bigger and bigger break-ins. Not long after the group was formed, the film *War Games* was released, and Landreth recalls that a flood of new, self-proclaimed hackers hit the bulletin boards and tried to join in the fun.[5] Predictably, most of the mainframe and minicomputer systems had major vulnerabilities. As a symptom of the problem, the designers of the first IBM PCs and compatibles thought that their systems would be safe if they were physically secure, i.e., the computers were in a locked room with a tamper-proof case or key entry. (See Figure 7.2.) These physical safeguards ignored the fact that computers were highly vulnerable when connected to other devices via the phone lines.

The Inner Circle exploited many of the vulnerabilities of networked computers, examining banking records, academic accounts, phone systems, and the reports of credit bureaus. It was a fascinating world of electronic transactions, secrets, and information. But on the afternoon of October 13, 1983, FBI agents entered the Landreth home near San Diego and seized Bill's computer equipment, telephone, and written notes. On the preceding day, federal agents had also visited the homes of nine other members of the Inner Circle in eight different states. Of particular interest was the group's use of GTE's Telenet network service.[6] Telenet was inspired by the protocols of the Arpanet, and it had hubs in 52 cities by the early 1980s. Landreth's equipment was eventually returned, but in May 1984 he was indicted for three counts of wire fraud by a federal grand jury in Alexandria, Virginia. He served a few days in jail, received an $87 fine, and was ordered to serve 3 years' probation. Bill Landreth was 18 years old at the time of his arrest.

Landreth's *Out of the Inner Circle* was published in 1985 to capture some of the excitement surrounding hacking in the national media, and to try to redirect Landreth and other hackers to more productive work that might benefit the

3. Information about connecting to computers through phone lines had been circulating in popular culture since an influential article on phone phreaking published in *Esquire* magazine; see Ron Rosenbaum, "Secrets of the Little Blue Box," *Esquire*, October 1, 1971, 117–124, 222–227.

4. Landreth, *Out of the Inner Circle*, 15.

5. Landreth, *Out of the Inner Circle*, 18, 35.

6. Landreth, *Out of the Inner Circle*, 208.

Figure 7.2 **An original IBM Personal Computer (1981) with the Mead-Hatcher Inc.'s locking "Fortress" cabinet, designed to secure a PC from unauthorized entry. In the early years of personal computing, securing a computer physically was considered an appropriate way to protect the device from unauthorized use.** (Courtesy of the Computer History Museum)

computer community. Just as *War Games* ultimately produced a happy ending (the teenager Matthew Broderick helped the government avert nuclear war), *Out of the Inner Circle* tried to show how a recalcitrant hacker might use his programming knowledge for good—in this case, how to help modern corporations manage security threats in the vulnerable world of electronic communications. The Microsoft Press marketing copy for *Out of the Inner Circle* emphasized the second point: "Estimates of losses from computer crime range from $100 million to more than $45 billion. No one knows the true extent of this epidemic—and we are all potential victims. How can *you* protect yourself and your company's data?"[7] (See Figure 7.3.)

The story seemed to end well, yet Bill Landreth's personal journey did not conclude so agreeably.[8] Although Landreth served only a brief time behind bars, he was

7. Landreth, *Out of the Inner Circle*, back cover.

8. See Matt Novak, "The untold story of the teen hackers who transformed the early Internet," *Gizmodo*, April 14, 2016.

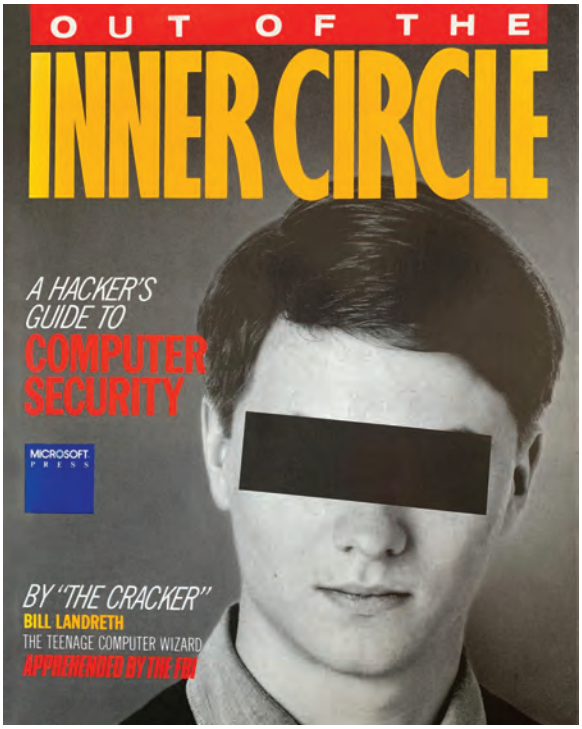


Figure 7.3 **Bill Landreth's *Out of the Inner Circle* (1985) was one of the first hacker biographies of the PC era that sympathetically portrayed a teen arrested for computer-related crimes.** (Used with permission from Microsoft)

tried as an adult, and after his conviction his road to rehabilitation was challenging. The teenager's parents moved to Alaska, eager to avoid the spotlight and to attend to their own troubles. But Landreth needed to stay in the area until his probation was complete. His local ties fell apart, and Landreth soon fell into depression and drug use.[9] He attempted to move to Mexico and then Oregon to make a new start, but he was caught violating his probation, and then forced to return to San Diego to serve 3 months in jail. Since that time, Landreth has moved around Southern California sporadically, but he has essentially been homeless for 30 years. His book agent, William Gladstone, arranged for the *Out of the Inner Circle* book project by pairing Landreth with an experienced co-author named Howard Rheingold. However, the money from this project was soon spent and Landreth has had difficulty finding steady work. The former hacker's last known contact was a 2016 interview with the journalist Matt Novak, a writer who has become interested in what became of Landreth and his associates. Novak found Bill Landreth living on the streets in

9. Matt Novak, "Teen hackers."

Santa Monica with a Samsung tablet and two large bags in his possession. Landreth indicated that he receives regular Social Security payments and California food stamps, but otherwise he has no regular source of income or support. Bill hasn't lived in a stable home since high school.

The activities of Landreth and the Inner Circle inspired a complete overhaul in how computer crime was prosecuted in the U.S. An August 1983 hacking case involving a different group of teenagers hacking into the Los Alamos National Laboratory in New Mexico raised similar concerns.[10] In 1984, the nation's first anti-hacking laws were established, in part to address concerns about protecting companies from unauthorized access as the Inner Circle and others had done. In the final report for his 1983 case, Landreth's crime boiled down to the fact that he had made three illegal long-distance phone calls without paying for them. Nothing was stolen, but the penetration of commercial computer networks was considered a serious crime at the time.[11] As the 1980s wore on, *illegal trespass* continued to be the biggest worry of government officials who monitored online activity. However, the members of professional computer organizations like the Association of Computing Machinery (ACM) were divided over the seriousness of the issue, and whether the blame was to be left at the feet of the teenage intruders or the information technology (IT) administrators who allowed the systems to be so vulnerable.[12] This debate essentially summed up the controversy about hacking in the mid-1980s, as the PC era gained momentum.

7.2 Jude Milhon: From Civil Rights Activist to Cyberpunk

All 15 members of the Inner Circle hacking community were male. The composition of this group (mostly teenage boys) aligns well with gender stereotypes about computer programming in the early days of PCs, and these tropes were applied even more strongly to hackers than to members of the general programming community. But there are important exceptions to the male hacker stereotype of the 1980s and 1990s, and the topic deserves much more attention by historians. One important example of a self-described female hacker and cyberpunk was Judith Milhon (1939–2003), commonly known as St. Jude later in life. Milhon was a self-taught programmer who became an important advocate for civil rights, as well as a writer, editor, and advocate for women in computing.

10. A summary of this story can be found in Joseph B. Treaster, "Trial and error by intruders led to entry into computer," *New York Times*, August 23, 1983, A1.

11. Matt Novak, "Teen hackers."

12. For an insightful discussion of the latter concern, see Rebecca Slayton, "Framing computer security and privacy, 1967–1992," in *Communities of Computing: Computer Science and Society in the ACM*, ed. Thomas J. Misa (San Francisco, CA: Morgan & Claypool/ACM Books, 2017), 287–329.

In technology circles, Milhon was an important member of the early computing community in the San Francisco Bay Area, where she contributed to BSD Unix, Community Memory projects, and *Mondo 2000* magazine. Steven Levy briefly noted Milhon's connection to the Community Memory project in *Hackers: Heroes of the Computer Revolution*, but to date there has been no comprehensive study of her life or influences.[13]

Judith Elaine Milhon was born on March 12, 1939 in Anderson, Indiana. She attended schools in Indiana and married Robert A. Behling in 1961, taking his last name. Attracted to the nascent countercultural movement, Judith Behling moved near Antioch College in Yellow Springs, Ohio, and established a communal household there with her husband, young daughter, and a growing collection of friends. Antioch College had long been associated with abolitionist movements and social activism, and in the mid-1960s it drew activists, hippies, and intellectuals from across the region.[14] In March 1965, Behling and her friends decided to engage more directly with the civil rights struggle, and they participated with Martin Luther King, Jr in the landmark voting rights march from Selma to Montgomery, Alabama. The protests gained momentum, and in the following month Behling was arrested for trespassing in Montgomery. On her intake form Judith was listed as a "housewife" from Yellow Springs, Ohio. Judith was interesting enough to the authorities that her mug shot and personal details were recorded in a Montgomery Police Department book compiled for local law enforcement officials (*Individuals Active in Civil Disturbances*, 1965).[15] A little later, Behling was arrested for civil disobedience in Jackson, Mississippi, and after this arrest she served jail time.[16] Behling's activist run-ins with the authorities greatly influenced her views and they solidified her reputation as an advocate for civil rights and free speech. The arrests also made her wary of public photographs, and for this reason it is hard to find additional images of her in archival materials, newspapers, or the records of computer companies.

13. A few episodes from Milhon's life in the 1970s can be found in Levy's *Hackers*, 160–161, 222, 282, and 285. For an obituary, see Sean Dodson, "Obituary: Judith Milhon: Making the internet a feminist issue." *The Guardian*, August 8, 2003, 27. A more personal reflection is available in Jon Lebkowsky, "Official bio of St. Jude," *The Well*, August 1, 2003. https://people.well.com/conf/inkwell.vue/topics/190/St-Jude-Memorial-and-Virtual-Wak-page01.html. Accessed August 29, 2019.

14. Lebkowksy, "Official bio of St. Jude."

15. State of Alabama, Department of Public Safety, Investigative and Identification Division, "Individuals Active in Civil Disturbances," Vol. 1 (1965). For the striking image of Behling, dated April 21, 1965, see http://digital.archives.alabama.gov/cdm/ref/collection/photo/id/1402.

16. Dodson, "Obituary: Judith Milhon," 27.

Judith Behling started programming in 1967 after reading a book on FORTRAN. The book was likely one of the popular FORTRAN primers published as the high-level language grew in popularity in the 1960s.[17] After learning the basics, Behling took a job as a programmer for the vending machine firm Horn & Hardart in New York City. Horn & Hardart ran the popular Automat restaurants in New York and Philadelphia, which dispensed pie, coffee, sandwiches, and hot food through coin-operated vending machines. Horn & Hardart were computerizing their systems and Milhon would have learned about their hardware, software, and data processing operations.[18]

In 1968, Judith Behling moved to the San Francisco Bay Area with long-time friend Efrem Lipkin, and her daughter, Tresca Behling. Judith separated from her husband and a divorce was granted by California courts in 1970. After this, Judith went by the name of Jude Milhon. Soon after arriving in California, Milhon worked at Berkeley Computer Company (BCC), an outgrowth of Project Genie, and she helped to implement the communications controller of the BCC timesharing system.[19] The components of this innovative computer can be studied through the writings of Butler W. Lampson, one of the founders of BCC who designed the system and later directed research at the Xerox Palo Alto Research Center.[20] Milhon thrived in the Berkeley area and soon met other people who shared her interests in computing and activism. During the Summer of 1968, Milhon met Lee Felsenstein, a local engineer and long-time political activist, and Milhon soon introduced Felsenstein to Efrem Lipkin, the man who traveled with her to California and became her long-term partner. According to Steven Levy, Lipkin was also a computer wizard—one of the most skilled programmers in that part of California.[21]

In 1971, the three partnered with other local activists and technologists at Project One, a high-tech commune of sorts in San Francisco that occupied space in an abandoned candy warehouse. Although Project One was organized into several fascinating subgroups (filmmakers, artisans, sculptors, and the like), Milhon, Felsenstein, and Lipkin were attracted to Resource One, a venture designed to build

17. For example, Daniel McCracken's *A Guide to FORTRAN Programming* (1961), or McCracken's *FORTRAN with Engineering Applications* (1967). For additional examples and the roots of FORTRAN, see Chapter 3.

18. I thank Ken Goffman for the detail about Horn & Hardart, which I received during a phone interview in July 2019.

19. Lebkowksy, "Official bio of St. Jude."

20. See Butler W. Lampson, "A Scheduling Philosophy for Multi-processing Systems," *Communications of the ACM* 11, no. 5 (May 1968): 347–359; Butler W. Lampson, "Some remarks on a large new time-sharing system." Internal memo, Berkeley Computer Corporation, September 1970.

21. Levy, *Hackers*, 160–161.

a people's computing center using an old Scientific Data Systems 940 time-sharing computer in San Francisco. Their goal was to create the region's first public computerized bulletin board system (BBS). Jude Milhon was not the only female programmer in the group; another self-described hacker and activist was Pam Hart, a woman featured along with several Resource One volunteers in a 1972 *Rolling Stone* article written by Stewart Brand.[22] Hart emphasized how political protest and programming tasks went hand-in-hand for the group during the tumultuous years of the civil rights movement:

> Then during the Cambodia Invasion demonstrations in Berkeley a group of us got together and designed a retrieval program for coordinating all of the actions on campus. It was a fairly dead system, but... it brought together people who had never worked together before and started them talking and thinking about how it was actually possible to do something positive with technology when *you* define the goals.[23]

Predictably, Jude Milhon was nowhere to be seen in the photos for the *Rolling Stone* article. It is a safe bet that she was nearby, however.

In 1973, a subset of the Resource One group broke away and partnered to launch the Community Memory project in Berkeley, a social networking experiment first introduced in Chapter 2. Community Memory was started by Lee Felsenstein, Efrem Lipkin, Mark Szpakowski, Ken Colstad, and Jude Milhon. The goal was to use computer time-sharing to establish electronic information hubs for the general public. By using a teleprinter and simple commands, novice users could compose short messages, associate them with keywords, and post them to the system for others to see. (See Figure 7.4.) Periodically, the group also produced a monthly index that listed the most recent entries by category. On a typical day, about 30 people made use of each of the terminals. During the 14-month trial period in 1973–1974, about 8,000 total entries were made on two public terminals.[24]

I examined the Community Memory Index for March 1974 in the Computer History Museum archive in Fremont, California, and it reveals an impressive range of topics, from requests for housing to health care to personal ads to music. Efrem

22. Stewart Brand, "SPACE WAR: Fanatic life and symbolic death among the computer bums," *Rolling Stone*, December 7, 1972, 50–56.

23. Pam Hart in Brand, "SPACE WAR," 56. See also Levy, *Hackers*, 165, for similar expressions of this sentiment.

24. Anon., "The Community Memory Project: An Introduction," advertising brochure (August 1982), 11. Available in Community Memory Records, Box 12, Folder 26, Computer History Museum Archive, Fremont, California.

Figure 7.4 **Community Memory terminal in Leopold's Records, Berkeley, California (1974). This was one of two early terminals used during the Community Memory trial program in 1973–1974.** (Courtesy of the Computer History Museum)

Lipkin posted that he was "Interested working with people to design and implement alternative forms of communication, learning, and subsistence. Especially interested in alternative economic forms."[25] An anonymous keyboard player simply wrote, "Keyboardist seeks any musicians who dig playing jazz, rock, or blues. We have gigs in Berkeley. Have bass player and harp player. Lots of experience."[26]

As Bo Doub of the Computer History Museum has written, Community Memory functioned as a pre-Web social network, a proto-Craigslist of the 1970s and 1980s.[27] A Community Memory pamphlet announcing the system indicates that the goal of the project was to revitalize the community through "strong, free,

25. Resource One/Community Memory Index (March 1974), p. 45 [Feb. 17, 1974]; in Community Memory Records, Box 13, Folder 5, Computer History Museum Archive.

26. Resource One/Community Memory Index (March 1974), p. 130 [undated]; in Community Memory Records, Box 13, Folder 5, Computer History Museum Archive.

27. Bo Doub, "Community Memory: Precedents in Social Media and Movements, Computer History Museum," Feb 23, 2016. http://www.computerhistory.org/atchm/community-memory-precedents-in-social-media-and-movements/. Accessed August 20, 2019.

non-hierarchical channels of communication—whether by computer and modem, pen and ink, telephone, or face-to-face."[28]

Jude Milhon was excited about the Community Memory project and its goals, but she was suspicious of hierarchy and she particularly disliked the male stereotypes associated with engineers and hackers in the Bay Area. Milhon once remarked after visiting a Home Brew Computer Club meeting in 1975 that there was a conspicuous lack of female hardware hackers, and that it was frustrating to see the male hacker obsession with technological play and power. The setting was summed up by Milhon with the epithet "[here are] the boys and their toys."[29] At Community Memory, Milhon worked against this exclusion and put her efforts into getting new, inexperienced users to experiment with the BBS. She did this by writing open-ended questions in the system about available resources in the region, such as "Where can I get a decent bagel in the Bay Area (Berkeley particularly)?"[30] More often than not, posts like this would get curious users to try out the system, and then longer conversations would ensue. Sharing information about what mattered to people on the fringe is what Jude cared about.

In the coming years, Milhon worked on BSD, a Unix-based operating system developed by the Computer Systems Research Group at UC Berkeley. There she worked with a range of Unix features and tools. When the Computer Professionals for Social Responsibility (CPSR) formed in 1983, Milhon joined the group and shared their concern about the potential use of computers in warfare. (Of major interest to CPSR was the U.S. government's support for the Strategic Defense Initiative—the so-called "Star Wars" missile shield project.) Milhon also turned to journalism as an outlet for her ideas about computing and social activism. In the mid-1980s, she contributed to the counterculture magazine *High Frontiers*, founded in 1984 by Ken Goffman, a skilled editor and writer who used the pseudonym "R. U. Sirius" in print. The magazine was based in San Francisco and it had a strongly counterculture vibe. For example, it creatively explored the local fringe community on topics related to technology, drugs, sex, and social issues. In 1988, the magazine changed its name to *Reality Hackers*, a title that more clearly emphasized the growing synthesis among programming, hacking, and psychedelic cultures. Milhon formally joined the magazine as Editor-in-Chief, taking the pseudonym "St. Jude" to emulate the public posture of the leading editors and contributors. In 1989, the

28. "The Community Memory Project: An Introduction," 1982, Community Memory records, Computer History Museum, Box 12, Folder 20, Catalog 102734414.

29. Milhon quoted in Levy, *Hackers*, 222.

30. Claire L. Evans, *Broad Band: The Untold Story of the Women Who Made the Internet* (London: Penguin, 2018), 102.

magazine reorganized, changed its name to *Mondo 2000*, and St. Jude became an associate editor and contributor of interviews and essays. She also spent a lot of time in the emerging online world of modems, bulletin boards, and distributed computing.

7.3 *Mondo 2000* and *The Cyberpunk Handbook*

Mondo 2000 was an astonishing publication for its era, helping to give birth to a creative expression called *cyberpunk culture*, a futuristic, science fiction aesthetic that interposed hacking, high technology, drug use, sex, anarchy, and goth sensibilities.[31] In a 2010 retrospective on *Mondo 2000*, the magazine's editors described their content mashup as "a tale of early digital culture, drugs, sex, surrealism, gonzo anthropology, death, digital culture, media hype, conspiracy paranoia, celebrities, transhumanism, irresponsible journalism, appropriation, hackers, pranks, theft, fun and desktop publishing."[32] Regular contributors to the magazine included science fiction writers and cultural critics like William Gibson, Timothy Leary, Maerian Morris, Rudy Rucker, Bruce Sterling, and Robert Anton Wilson. The vivid design and literary format of *Mondo 2000* contributed to the aesthetic that would eventually find popular expression in *Wired* magazine, which debuted in 1993 and featured some of the same writers. (See Figure 7.5 for the Summer 1990 issue of *Mondo 2000*.)

Ken Goffman's early influence on the *Mondo 2000* team was considerable. Goffman wrote hundreds of creative essays and 10 books, including two collaborations with Timothy Leary. He was fascinated with the concept of *cyberspace*, a term that could be abridged as an immersive artificial world of information that could be entered, explored, and manipulated electronically.[33] This is the definition that the *Mondo 2000* editors provided *Time* magazine in early 1993, when the trade publication ran a feature story on cyberpunks and the early commercial Internet. In terms of wiring, *Time* defined the new infrastructure as "the globe circling, interconnected telephone network that is the conduit for billions of voice,

31. For an introduction to the term *cyberpunk* and its histories, see Steven Levy, *Crypto: How the Code Rebels Beat the Government—Saving Privacy in the Digital Age* (New York: Viking, 2001); *High Noon on the Electronic Frontier: Conceptual Issues in Cyberspace*, ed. Peter Ludlow (Cambridge, MA: The MIT Press, 1996).

32. David Pescovitz, "Mondo 2000: An Open Source History," *bOING bOING*, April 6, 2010. https://boingboing.net/2010/04/06/mondo-2000-an-open-s.html. Accessed on August 20, 2019. The world of *Mondo 2000* can also be surveyed in the volume by Rudy Rucker, R. U. Sirius, and Queen Mu, *Mondo 2000: A User's Guide To The New Edge: Cyberpunk, Virtual Reality, Wetware, Designer Aphrodisiacs, Artificial Life, Techno-Erotic Paganism, and More* (New York: Perennial, 1992).

33. This is but one of many descriptions. For an earlier use of the term, see William Gibson, *Neuromancer* (New York: Ace, 1984).

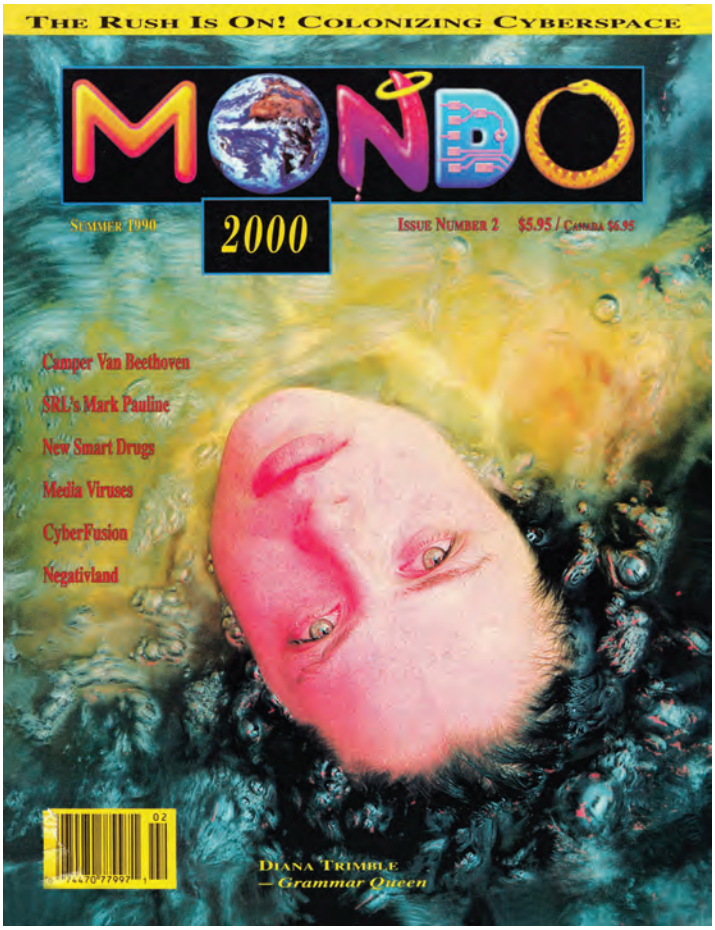


Figure 7.5 **Issue #2 of *Mondo 2000* (Summer 1990). Photo and cover design by Bart Nagel.** (Used with permission of Bart Nagel)

fax and computer-to-computer communications."[34] In the same article, *Time* estimated that 17.5 million Americans were currently able to connect through modem-equipped computers, bulletin boards, and online services such as GEnie, Prodigy, and CompuServe.[35]

The American public was relatively inexperienced with this world, but Bay Area technologists had been experimenting with digital connectivity for decades. Goffman was assisted by Alison Bailey Kennedy ("Queen Mu"), a Palo Alto insider who served as the Editor-in-Chief of *Mondo 2000* after its initial startup period. Kennedy

34. Philip Elmer Dewitt, "Cyberpunk!" *Time*, February 8, 1993: 60.

35. Dewitt, "Cyberpunk!" *Time*, 60.

attended Palo Alto High School and studied anthropology in graduate school, publishing articles on Mesoamerican iconography and the hallucinatory effects of toad and tarantula venom.[36] She met Ken Kesey and the Merry Pranksters in the early 1960s, and she later worked part time at the Portola Institute in Menlo Park. Together with Bart Nagel, a talented graphic artist and photographer, Kennedy and Goffman set about building an audience for their oversized, wide-ranging magazine. In September 1990, Alison Bailey Kennedy was quoted in the *New York Times* saying that *Mondo 2000* had a subscriber list of 4,000 customers, with additional sales of 25,000 copies per issue in newsstands.[37] These figures offer a preliminary indication of the expanding influence of cyberpunk culture in American society. Kennedy explained the growth in a more comprehensive way: "We're dealing with the cultural manifestations of the P.C. revolution."[38]

St. Jude's contributions to *Mondo 2000* were also significant. She first contacted people in her personal and professional networks to write articles for the magazine, including Lee Felsenstein. His first essay described what the ultimate cyberpunk PC might look like, an allusion to the "Tom Swift Terminal" that Felsenstein prototyped in 1974.[39] (For more information on this influential device, see Chapter 2.) St. Jude was also a regular interviewer of high-tech personalities and theorists, gathering opinions about what seemed new and interesting in high-tech counterculture. Her byline acknowledged her formative influence on the hacker communities of the 1970s and 1980s. It also mentioned that she had worked as a physician's assistant in the Bay Area, part of her interest in health care that stretched back to the early 1970s.[40] In 1991, St. Jude was promoted to Managing Editor of *Mondo 2000*, and she began a regular column entitled "Irresponsible Journalism," which became her most recognizable publishing platform. The segment presented her evolving ideas on cyberpunk culture, sexuality, health care, and technology.

In 1995, St. Jude co-authored a book with longtime magazine collaborators Ken Goffman and Bart Nagel entitled *The Cyberpunk Handbook*.[41] (See Figure 7.6.) This compendium was a kind of visual almanac for cyberpunk culture, an aesthetic

36. The information that follows comes from an interview that I conducted with Alison Bailey Kennedy on July 11, 2019. For additional information, see Lawrence M. Fisher, "Style makers; Ken Goffman and Alison Kennedy, magazine editors," *New York Times*, September 29, 1990, Section 1, 40.

37. Fisher, "Style makers," *New York Times*, 40.

38. Fisher, "Style makers," *New York Times*, 40.

39. Lee Felsenstein, "The cyberpunk computer," *Mondo 2000* 2 (Summer 1990): 21.

40. See "Contributors," *Mondo 2000* 1 (Fall 1989): 159.

41. St. Jude, R.U. Sirius, and Bart Nagel, *The Cyberpunk Handbook: The Real Cyberpunk Fakebook* (New York: Random House, 1995).

Figure 7.6 ***The Cyberpunk Handbook*** **(1995), by R. U. Sirius [Ken Goffman], St. Jude [Jude Milhon], and Bart Nagel.** (Used with the permission of Penguin Random House LLC)

that included science fiction, goth fashion, leftist politics, drug culture, and other anti-establishment norms. Unlike Bill Landreth's hacker treatise, *The Cyberpunk Handbook* contained few insights about cracking corporate portals or telecommunication hubs. Instead, the book described the language and visual imagery of the new subculture, including chapters about new terminology, acronyms, cyberpunk fashion, establishing an online persona, and essential books, films, and video games. Through photographs and illustrations, the authors presented model forms of cyberpunk dress as a "semiosis of black leather, chrome, mirrorshades, and modems."[42] Ramen noodles and Jolt Cola were recommended as "the haqr

42. St. Jude et al., *Cyberpunk Handbook*, 188.

[sic] staffs of life."[43] The first commercial browsers of the Internet era were also downgraded as disappointing "four-star hotels that will likely kill true hacking." Instead, the authors recommend Unix (St. Jude's preferred operating system), where true cyberpunks were welcomed into hacker culture via "a dark endless maze of catwalks and mantraps, an eternal hard-hat area that kills the foolish and shelters the brave."[44]

Gender is also a revealing category in *The Cyberpunk Handbook*, and the authors portrayed a range of styles and behaviors that could differentiate male and female cyberpunks. Occasionally these schemes worked to undermine or intermingle established stereotypes. For example, women and girls were prominently featured in photographs, often wearing gothic clothing with visible piercings, jewelry, lit cigarettes, and an assortment of high-tech gear (flip phones, modems, headsets, and iconic "laser pointers"). Although sexuality is discussed overtly, the authors hinted that they were self-censoring true cyberpunk culture because Random House would not permit more revealing subjects or attire. A few aesthetic descriptions from the book will suffice: "The standard cyberpunk costume is ideal for riding motorcycles, and a mirror-shades helmet is a big plus for the cyber look…" "Goths, deathcore, and vampire-wannabes… you should know about The Cure, which is a band. To fit in, grow your hair big and dye it blue-black."[45]

Sometimes the recommended outfits were gender inclusive. For example, "all sexes should wear a Victorian shirt-blouse—white or black only—that gapes to show flesh. You must practice looking tormented, tall and thin."[46] Girls and women were described with pliable language, such as alternative capitalizations and eclectic spellings for common words. For example, the term "Riot Grrrls," is used to describe "fierce girls who like tech." The authors also conceded, "This is a sexist category, but there we are: girls only. A grrl can be called 'd00d' and 'guy' at all times, but a non-female guy is not a grrrl. This is just the way things are."[47] In short, *The Cyberpunk Handbook* attempted to push the boundaries of computing culture into new areas. One of the goals seemed to be female empowerment. However, promiscuity and open sexuality was also a part of the culture, a fusion between 1960s counterculture values and the punk rock norms of a new generation. The mashup of attitudes about gender is expressed in the following statement: "If you're a grrrl, you can wear anything you want to, because you're there to defend it… NOTHING is more

43. St. Jude et al., *Cyberpunk Handbook*, 66.

44. St. Jude et al., *Cyberpunk Handbook*, 72.

45. St. Jude et al., *Cyberpunk Handbook*, 30.

46. St. Jude et al., *Cyberpunk Handbook*, 30.

47. St. Jude et al., *Cyberpunk Handbook*, 31.

attractive than a fierce, blazing, ninja-type grrrl right now, and if she knows UNIX or phone-freeking, the world is hers. Hrrrs."[48]

7.4 Cypherpunks and Cryptography

In 1992, St. Jude began meeting with a group of Bay Area programmers that became particularly fascinated with cryptology and cryptography. These were hackers and libertarians who had deep concerns about government surveillance and the security of information transmitted in online environments. The group included Timothy C. May, Eric Hughes, and John Gilmore. Soon it grew to include hundreds of people—both inside and outside of the U.S. The cohort became especially interested in data security issues. Fundamentally, they hoped to use cryptography and privacy-enhancing software to enact social and political change.[49] In a world replete with online transactions, they feared government monitoring and interference. As private citizens, however, they argued that individuals had a right and responsibility to protect their digital privacy and to ensure that personal electronic information was kept safe.[50] As a route to enable these protections, they discussed the best ways to use secret codes and ciphers to encrypt personal data so that it could not be read by prying eyes. Like other cryptographers, they discussed productive ways to use ciphers to scramble files into ciphertext (a scrambled message), and then how to decrypt (or reconstruct) encrypted information so that it could become readable again. In some ways, the group was ahead of government regulators and politicians, who had only recently formulated laws to govern how digital information should be protected in online contexts. As online transactions became more prevalent in the early 1990s, the issue became a matter of concern for government regulators as well as businesses and regular citizens. Everyone worried about online privacy, the threat of hackers and hacking, and how sensitive information might be transferred across national borders.

In a September 1992 meeting of the cryptography group in Berkeley, Jude Milhon coined the term *cypherpunk* to describe the collective activities of the hackers, programmers, and mathematicians who were working to promote online privacy.[51]

48. St. Jude et al., *Cyberpunk Handbook*, 31. "Phone-freeking" is a term that means hacking over phone lines.

49. Nathaniel Popper, "Timothy C. May, early advocate of Internet privacy, Dies at 66," *New York Times*, December 21, 2018.

50. Timothy C. May, "The Crypto Anarchist Manifesto," November 22, 1992. https://www.activism.net/cypherpunk/crypto-anarchy.html. Accessed on August 20, 2019.

51. Levy, *Crypto*, 211.

Milhon had been invited to the meeting by her friend Eric Hughes, and she was energized by the event. Milhon suggested that the term cypherpunk was a combination of the words *cypher* (or *cipher*) and *cyberpunk*. The group responded enthusiastically to the appellation, and cypherpunk became the privacy movement's official name in the years to come.[52] After 1992, *cypherpunks* also became the name of the privacy group's listserv (or electronic mailing list), a grouping also organized by Eric Hughes.[53]

As the cypherpunks evolved, the hackers developed encryption algorithms and software applications to promote online privacy. They also wrote position papers on the importance of cryptography, and forcefully advocated for the reduction of government restrictions on encryption. Eric Hughes 1993 treatise, *The Cypherpunk Manifesto*, emphasized the skills that cypherpunks needed. "Cypherpunks write code," Hughes explained.

> They know that someone has to write to defend privacy, and since it's their privacy, they're going to write it. Cypherpunks publish their code so that their fellow cypherpunks may practice and play with it.[54]

In plain terms, the cypherpunks sought to maintain their identity as computer programmers and problem solvers, and to use their coding skills to develop workable encryption and privacy solutions. The founders of the movement also had serious programming chops. Timothy May was a professional physicist and engineer at Intel, and John Gilmore was an early employee at Sun Microsystems who made substantial contributions to The GNU Project. Eric Hughes was also a successful mathematician and software developer, at home with the intricacies of cryptography and software creation. The three garnered national attention in May 1993, when *Wired* magazine dedicated an issue to cryptography and the cypherpunks. The magazine's cover depicted the three men in masks, indicating their desire for privacy but also their marginalized status as would-be hacker criminals. Steven Levy, the author of the featured essay, continued to popularize the cause of cypherpunks with his book *Crypto: How the Code Rebels Beat the Government — Saving Privacy in the Digital Age* (2001).

52. St. Jude et al., *Cyberpunk Handbook*, 45.

53. Robert Manne, "The Cypherpunk Revolutionary: Julian Assange," *The Monthly*, March 2011. https://www.themonthly.com.au/issue/2011/february/1324596189/robert-manne/cypherpunk-revolutionary. Accessed on August 19, 2019.

54. Eric Hughes, "A Cypherpunk's Manifesto," March 9, 1993. https://www.activism.net/cypherpunk/manifesto.html. Accessed on August 19, 2019.

Despite their masked and/or marginal status, the hackers, cyberpunks, and cypherpunks that I have been discussing in this chapter deserve recognition as members of America's programming community, the multifaceted entity that I have been describing as Code Nation in this book. Regardless of the risks that hackers, cyberpunks, and cypherpunks may have posed to the computing infrastructure, their actions helped mainstream users think about emerging issues related to computer security, privacy, online access, and digital citizenship. These were important subjects as computing grew from a niche engineering practice into a ubiquitous technology that influenced many aspects of life. Hackers often mirrored the anxieties of mainstream computer users, pointing out who held insider status and who was being pushed to the margins. Hackers and cyberpunks were not just threatening, however; the computing elites could also read *Mondo 2000* and *Wired* magazine, and thereby participate in a culture that they found fascinating and alluring, while remaining safely embedded in more conservative institutions. Jude Milhon's liminal role as a magazine editor was especially important in this regard. She offered glimpses of a cyberpunk culture in which sex, hacking, drugs, and coding all blended together—a notion not readily accepted by mainstream society. St. Jude regularly traversed the boundaries of numerous personal and technical identities, including software developer, political activist, health-care worker, journalist, hacker, mother, and cypherpunk. As in other times and places, the marginalized were indispensable for polite society; the center and periphery existed in relationship as two sides of the same coin.[55]

It is challenging to recover the voices of people on the margins of society, because they are not always recorded by traditional narratives. Restoring hackers and cyberpunks to the history of programming and personal computing will restore some of this balance. In the next chapter, we'll search for additional sources of information about the users of computers via computer trade magazines, printed periodicals that played an important role in how America's computing societies interacted and took shape in the 1980s and 1990s. Computer magazines diffused valuable information about using and programming PCs in the era of MS-DOS, Unix, the Macintosh, and Windows. But periodicals were also two-way streets, with protests and commentary arriving from users on a monthly basis through "letters

55. On the reciprocal relationship between dominant and marginal cultures, historian Barbara A. Hanawalt has written, "Indeed, a symbiotic relationship often existed between marginals and the social establishment. The dominant culture needed the services of marginals for their own purposes. The mainstream found the margins a place for thrills and titillation—a place to live dangerously—to have sex, to engage in petty crime, or to commit major fraud… the exemplars of bad behavior were very useful for setting off good behavior." Hanawalt, *Living Dangerously on the Margins*, 2.

to the editor" columns, "how to" articles, essays, and product reviews. We'll sample the richness of this data, and see how it might be used to chart the experiences of novice computer users, power users, and professional programmers. We'll also consider how computer users accepted, accommodated, or rejected the dominant viewpoints of industry elites.

8 Computer Magazines and Historical Research

"Sure, modern PCs are more powerful than ever. But the PC revolution was about freedom, not power… There is no doubt that network management is necessary. Let's just be sure that control of the machines doesn't mean restricting the creativity of their users."

Alun Whittaker, *Byte* (April 1991)

"As a hacker and a programmer, I am completely appalled at the lack of respect that computer programmers, users, and hackers get. Face it, folks, hackers are actually a good thing."

Stephen Bobic, *Macworld* (June 1991)

The early 1990s brought a surge of interest in personal computing, with the MS-DOS, Macintosh, Windows, OS/2, and Unix platforms all attracting followers in the commercial marketplace. America's media companies were quick to capitalize on the growing interest in home and business computing, and the number of technical books and magazine publishers skyrocketed as the new emerging platforms gained momentum. Although the commercial Internet would eventually become the conduit through which marketing and technical information flowed, the early 1990s were still bathed in the golden light of print publishing, a 500-year-old industry that produced the lion's share of computer training materials through the early 2000s. (For a summary of how the Internet disrupted book and magazine publishing, see the Afterword.) Throughout the 1980s and 1990s, the ranks of Code Nation swelled, and when they needed technical information about how to use their computers, they often turned to books, magazines, journals, and newsletters. On occasion, these publications also led to face-to-face encounters with fellow users, at special interest group meetings (SIGs), industry trade shows, computer classes, or other gatherings.

Chapters 3, 4, 5 and 6 introduced computer book publishing as a lively source of information about America's programmers and computer users. This chapter explores magazine publishing as a second window into personal computer (PC) culture, drawing attention to what historians can learn about "pre-Internet societies" if we take printed publications seriously as sources of information about the attitudes of typical computer users. I have followed an approach similar to that of scholars of 16th- and 17th-century Europe, using popular print materials to study the cultural and intellectual milieus of Renaissance and Reformation Europe. In this setting, historians have evaluated publications such as pamphlets, sermons, broadsides, almanacs, newspapers, and chapbooks to examine the mentalities of people as they experienced periods of change and consolidation associated with religious reformation and revolution.[1] An important part of this work has been uncovering the opinions of marginalized figures, such as day laborers, artisans, peasants, heretics, outcasts, and orphans. These are the historical agents that are sometimes overlooked as we train our gaze on the wealthy, well-educated, and powerful members of society.

Somewhat more recently, the historians of America's technical past have also experimented with this approach, finding in the electronic age's magazines, newsletters, and informal writings the hidden voices of technology users that have been glossed over by more traditional histories.[2] In this chapter, I plan to continue this work by using "letters to the editor" columns, "how to" articles, and a selection of product reviews to capture the enthusiasm and complaints of typical computer users as PCs came into regular use in America. These materials will provide insights into the experiences of computing novices, power users, and programmers. We'll also consider how computing audiences accepted, accommodated, or rejected the marketing strategies and product preferences of industry elites. I hope that this approach will be of methodological interest to future historians.

1. Representative works include Robert W. Scribner, *For the Sake of Simple Folk: Popular Propaganda for the German Reformation* (Cambridge: University of Cambridge Press, 1981); Elizabeth Eisenstein, *The Printing Press as Agent of Social Change* (Cambridge: University of Cambridge Press, 1993); Mark U. Edwards, *Printing, Propaganda, and Martin Luther* (Minneapolis: Fortress Press, 1994); and Andrew Pettegree, *The Invention of News: How the World Came to Know About Itself* (New Haven, CT: Yale University Press, 2014).

2. For a pioneering approach, see Carolyn Marvin, *When Old Technologies Were New* (Oxford: Oxford University Press, 1990). Recent studies that make creative use of newsletters and periodicals include Kevin Gotkin, "When computers were amateur," *IEEE Annals of the History of Computing* 36, no. 2 (2014): 4–14; and William F. Vogel "'The spitting image of a woman programmer:' Changing portrayals of women in the American computing industry, 1958–1985," *IEEE Annals of the History of Computing* 39, no. 2 (2017): 49–64.

By the early 1990s, there were hundreds of regular computer magazines and newsletters in print in the U.S. This chapter presents representative samples from some of the most popular publications, including *Byte*, *Communications of the ACM, Dr. Dobb's Journal*, *IEEE Computer*, *Macworld*, *PC/Computing*, *PC Magazine*, and *UNIX World*. I selected these magazines because they addressed many of the leading PC platforms and user groups in the early 1990s, but the selection is necessarily limited. My selection of eight trade magazines is only a sample of the rich collection of periodicals that currently slumber in America's technical libraries, awaiting visits from historians, computer scientists, sociologists, and other scholars of computing. By the early 1990s, the size of these publications stretched to 450 pages or more for each issue. At that time, the circulation rates for the leading high-tech magazines approached 500,000 copies per issue, indicating widespread use and impact.

Letters to the editors are fascinating because they reveal early adopters struggling with computer technologies and voicing support (or disdain) for emerging platforms and the companies that advocated for them. In an era before Internet information hubs—corporate websites, social media, Google Ads, and YouTube channels—many users turned to magazines, newsletters, and journals as their sources of support when they experimented with new technologies. Letters to the editor are fascinating interludes in these publications, because they were usually printed in prominent locations with replies from the editorial staff, who either defended their opinions or made common-cause with frustrated readers. As a result, this material allows historians to eavesdrop on conversations about new hardware and software as they were taking place in emerging computing communities. In the case of programming instruction, letters to the editor give us insight into how hobbyist and professional communities learned to code, and how they accepted, rejected, or altered the systems presented by software publishers. In short, we get month-by-month updates on the highs and lows of the learn-to-program movement as it entered its corporate and commercial manifestations. (For more about the stages in this transition, see the chapters in Part III.)

To recover the voices of computer users, I have surveyed hundreds of letters to the editor from the early 1990s, using bound magazine collections in engineering libraries as source material. Although some historic collections of this type remain in public institutions, many periodicals have been lost to time and corporate consolidation. The data in this chapter is thus qualitative and preliminary. However, the coding and recording process has revealed one important dynamic: magazines are disappearing from our libraries at an alarming rate, in the same way that popular tracts from the 17th century were discarded when elites assumed that they contained little of value. As noted in Chapter 1, computer books and magazines often seem transitory and ephemeral to library personal, especially once we have

transitioned from one technical system to the next. But these documents are precisely the sources that future generations will find fascinating, because they capture our real-world struggles and achievements with technology, including glimpses of America's past before ubiquitous computing took hold.

8.1 Magazines and a Popular Culture of Computing

Computer periodicals started in the 1950s in the U.S., when research on digital computing gained momentum and institutional support. Published in monthly or quarterly issues, computer periodicals slowly expanded into a range of technical publications in the 1960s, including *academic journals* (presenting peer-reviewed research), *industry journals* (covering trends in corporate computing and information processing), and *trade magazines* (offering computer-related information to the general public).

Trade magazines have been very durable sources of information in America. As new media has arrived over the past century (radio, film, television, computing), magazine publishers have quickly adapted to the offerings and worked to profit from them. For example, when television grew in popularity in the 1960s, the most popular periodical in the U.S. was the *TV Guide*, which its publishers believed would enhance the television-viewing experience by introducing new celebrities and marketing upcoming shows. The same thing happened when commercial computing arrived; the new digital media did not replace magazines, but magazine publishers adjusted their offerings to provide more information about how computers worked, including ads for new products. Magazines were also much faster to produce than books, and they offered more space for columns and information than traditional newspapers. Over time, trade magazine publishers developed a successful business model that devoted dozens (or hundreds) of pages to product ads, and they sold magazines directly to subscribers. In addition, they explored new ways to stock their products in retail stores, including the first computer stores. It is no wonder that in the history of personal computing, the first industry "experts" were actually the publishers of computer newsletters and magazines, such as Bob Albrecht's *People's Computer Company* (PCC) newsletter, and its successor, *Dr. Dobb's Journal of Tiny BASIC Calisthenics & Orthodontia*.

The first mass-market industry journals were *Computers and Automation* (1951), *IEEE Transactions on Computers* (1952), *Journal of the ACM* (1954), and *Communications of the ACM* (1958). (See Figure 3.3 for an early issue of *Communications of the ACM*.) These periodicals were professional sources of news and information from the first computer societies in America, and they gradually sought to fill the gap between trade-oriented news magazines and academic research journals, which were oriented around peer-reviewed research articles and notices about upcoming

academic conferences. One of the originators of this format was Edmond Berkeley (1909–1988), a co-founder of the Association for Computer Machinery (ACM), who edited *Computers and Automation*, a monthly journal that discussed news, technical developments in computing hardware, and developing social issues, such as the ethical responsibility of scientists for the innovations that they created.[3] As the years passed, professional computing societies developed dozens of magazines and journals to address the interest groups within their memberships. Gradually, trade magazines also emerged that took advantage of new print formats, photography, color, advertising, and the concerns of computer workers and associated fields.

In the era of microcomputers, the oldest computer magazines were *Creative Computing* (1974), published by David Ahl; and *Byte* (1975), published by McGraw-Hill. (For an early issue of *Creative Computing*, see Figure 11.2.) Also from this era was *Dr. Dobb's Journal* (1976), published originally under the title *Dr. Dobb's Journal of Tiny BASIC Calisthenics & Orthodontia* (see Chapter 3). Information about PCs and their forbearers was also published by the industry stalwarts *Hewlett-Packard Journal* (1949) and *Popular Electronics* (1954). *Popular Electronics* famously jumpstarted the PC era with its January 1975 cover story on the Altair 8800. What many modern computer users don't know is that the magazine had been a strong seller since the 1950s, regularly introducing new technologies to a world-wide audience. Ziff-Davis proclaimed as much when they advertised *Popular Electronics* as the "World's Largest-Selling Electronics Magazine." In the April 1957 issue, Ziff-Davis reported an average net paid circulation of 240,151 copies.[4]

By the late 1980s, most personal computing magazines were published monthly, with relatively low prices for annual subscriptions. (*Byte* was initially $10 per year.) Like newspapers, magazine publishers wanted to attract a wide audience so that they could sell advertising space to companies who wanted to market their products. For this reason, the most successful computer magazines tended to be large, with most of their space dedicated to advertisements. Ads were formatted as half-page, full-page, or multi-page commercials. Computer corporations advertised to spread the word about their platforms, but so did mail-order resellers, who accumulated and sold a wide range of hardware and software products. Resellers typically advertised their current price list and included an address or phone number where customers could specify quantities and arrange payment. These channels assisted

3. For a summary of Berkeley's ethical concerns about computing, see Bernadette Longo, *Edmund Berkeley and the Social Responsibility of Computer Professionals* (San Rafael, CA: Morgan and Claypool Publishers/ACM Books, 2015).

4. "Contents," *Popular Electronics*, April 1957, 4. Throughout the 1950s, the magazine often listed its average circulation figures on the contents page.

in making PC products a type of commodity that could be bought or sold without much interaction with dealers or account representatives.

For price-sensitive items like memory chips, toner cartridges, and peripherals, advertising pages provide insight into the dynamics of computer price wars, and they can also be used to reconstruct the total cost of ownership for computer systems, less the expense of installing and maintaining software. When the IBM PC "clones" arrived, computer magazines were filled with the technical details of the new systems and where they could be located. While PC clone makers continually lowered their prices and offered a wide range of options, Apple Computer encouraged retail customers to buy Macs from authorized retailers or directly from the company. For this reason, *Macworld* (1984) and other Mac-oriented magazines did not contain pricing information about the unit costs for Macintoshes, although they did present a steady stream of software options and third-party peripherals. (See Figure 8.1.) The end result of the pricing and merchandising wars was that consumers paid a little more for owning a Macintosh, but many felt that it was worth it.

Did the general public really notice when scores of new computer magazines arrived? A 1983 article in the *New York Times* ("Boom in computer magazines") makes it clear that *something* exciting was happening on the publishing front, as customers suddenly noticed the presence of many new magazines at local newsstands. Just 2 years after the introduction of the IBM Personal Computer, there were already 200 magazine series available.[5] The average monthly circulation rate was over 250,000 copies each for the top five magazines. *Computers and Electronics*, published by Ziff-Davis, enjoyed a paid circulation of 550,000, followed by *Personal Computing*, with 460,000; *Byte*, with 420,000; *Popular Computing*, with 306,000; and *Compute!*, with 270,000.[6] Of special note was the surge in computer advertising, which noticeably fattened many magazines. According to the *New York Times*, two of the top three magazines in terms of page count were computer publications, an indication of rising advertising revenue:

> *Byte* topped the list with an average page count of 543. *Brides*, published by Conde Nast Publications Inc., was second, with 410. *80 Micro*, published by Wayne Greene, whose computer magazine publishing company was acquired

5. "Boom in computer magazines," *New York Times*, November 9, 1983.

6. "Boom in computer magazines," *New York Times*, November 9, 1983. Figures supplied by Shelia Clark from Adscope, a marketing statistics company that conducted research on computer magazines.

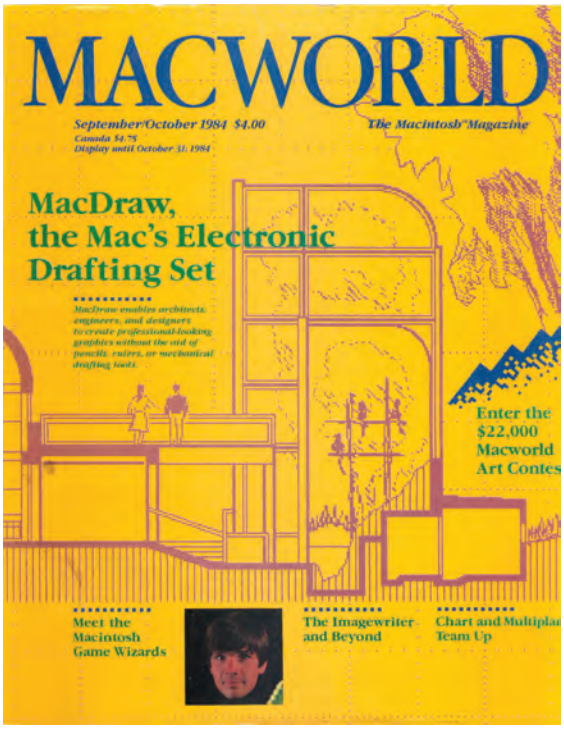


Figure 8.1 ***Macworld* magazine (September/October 1984).** (Used with permission of Macworld Copyright ©2019. All rights reserved)

last summer by a unit of the International Data Group in a $60 million agreement, was third, with 392 pages.[7]

One of the interesting developments in magazines and other periodicals was the focus on *niche publishing*, which encouraged the development of interest groups around specific platforms, applications, and subjects. For example, *Antic* magazine started in 1982 and ran to mid-1990. *Antic* supported the 8-bit family of Atari computers and included reviews, tutorials, and BASIC games that readers could type in directly from the pages. *Amiga World*, published by International Data Group (IDG) Publishing, was designed for the owners of Amiga PCs. It was published from 1985 to 1995, and catered especially to gamers who valued the platform's advanced graphics capabilities, sound, and video resources.

7. "Boom in computer magazines," *New York Times*, November 9, 1983. The information was provided by Jay Walker, publisher of *Folio 400*, a magazine industry publication.

The niche-publishing concept also applied to a growing cohort of newsletter publishers, who focused less on advertising and retail sales, and more on custom content for professionals who would pay extra for insights into their industry. For example, the Cobb Group's *Inside Microsoft BASIC* was a monthly newsletter devoted to BASIC and QuickBASIC programming techniques for hobbyists and aspiring professionals. (I wrote several articles for this newsletter in the early 1990s, as did other computer book authors.) *P. C. Letter: The Insider's Guide to the PC Industry* was a well-respected newsletter edited by Stewart Alsop II, a regular speaker and prognosticator in the software industry. Swiss-born journalist and businesswoman Esther Dyson also published several influential newsletters of this type, including *Release 1.0*, a monthly technology report that began in 1983. O'Reilly took over publication of this newsletter in 2006, and as a boon to computer historians, they currently provide access to all the back issues of *Release 1.0* for free via the Web.[8]

Stretching even farther back, the original executive newsletter in commercial computing was perhaps *The Seybold Report*, created by John W. Seybold and his son, Jonathan, in 1971. *The Seybold Report* was essentially a high-cost news bulletin published twice a month, with commentary and predictions for publishing professionals who were navigating the process of computerization in their industry.[9] Executives highly valued the personal nature of this report, despite its cost, and they used it to predict emerging technology trends and their consequences. Newsletter recipients hoped to receive inside information weeks or months before their competitors.

Finally, a word on marketing and the commercial impact of computer periodicals. As computer magazines and newsletters become more established in the mid-1980s, they became locations for hardware and software publishers to establish brand identities.[10] Magazines were places to browse advertisements, read product reviews, seek the advice of industry experts, and consider helpful "tips, tricks, and traps" about emerging platforms. When a major software release came out, such as MS-DOS 5.0 (available June 1991), all the trade computer magazines ran feature articles on the platform and tried to outdo each other with their coverage. (See Figure 8.2.) Magazine subscriptions and advertising revenues also surged when new products were released. For example, in September 1991, *PC Magazine* announced that it had a monthly circulation of 800,000 copies, and that 83% of

8. For more information, visit http://radar.oreilly.com/r2/release1-0. Accessed August 21, 2019.

9. John Markoff, "John W. Seybold, 88, innovator in printing," *New York Times*, March 16, 2004.

10. Quint Randle, "A historical overview of the effects of new mass media introductions on magazine publishing in the 20th century," *First Monday*, Vol. 6, no. 9, September 3, 2001. https://firstmonday.org/article/view/885/794. Accessed August 12, 2019.

Figure 8.2 ***PC Magazine*** **(July 1991).** (Reprinted with permission. ©2019 Ziff Davis, LLC. All Rights Reserved)

these readers were in charge of buying computer hardware or software. Underlining the importance of this marketing vehicle, *PC Magazine* also announced that its advertisements had generated over $200 million in sales revenue for advertisers that year.[11] The conclusion they wanted readers to draw was that *PC Magazine* was the premier branding platform for the PC industry, as well as an important source of information for users and the companies they worked for.

8.2 Letters from the Programming Community

Now to the letters. As a preliminary effort to study the reactions of users to their software and hardware platforms, I present here a representative sample of reader comments from *Byte*, *Computer*, *Communications of the ACM, Dr. Dobb's Journal*, *Macworld*, *PC/Computing*, *PC Magazine*, and *UNIX World*. Rather than a quantitative analysis, I offer a qualitative reflection organized around several user categories

11. Sales advertisement, *PC Magazine*, Sept. 24, 1991, 480.

and themes. The letters come from the year 1991, which I selected because it represents an important milestone in PC platforms and the software releases for MS-DOS, Microsoft Windows, and the Apple Macintosh. (For more about the products released during this year, see Chapter 6.)

Of the letters that I surveyed, approximately 95% were written by users with traditional male names, and 5% were written by users with traditional female names.[12] This disparity fits the general pattern of gender imbalance noted in several recent studies of computer cultures in the 1980s and 1990s in America. It may also support the observation that women's contributions to technical fields are often made invisible by cultural expectations, such as who should consider publishing their opinions in a mass-market technical magazine.[13] In terms of location, however, the letter writers hailed from a variety of places in the U.S., and their home city is presented when available. To tease out a range of experiences and expertise, I have organized the letters into the following socio-technical categories, which attempt to assess the technical proficiencies that the letter writers may have possessed. These categories appear as headings in the following sections, where I also provide contextual information.

- New PC users (those looking for advice about first steps with PCs or worrying about complexity).
- Power users (those who maintained PCs, installed new software, and composed and edited batch files, but rarely wrote programs).
- Advanced hobbyists (proficient power users, hackers, and gurus that also mention programming experiences with their PCs, typically using BASIC, Pascal, or C).
- Professional programmers (advanced software developers who have professional work experiences using Pascal, C, assembly language, and other tools; these programmers often reveal advanced expertise with specific platforms, such as MS-DOS, Mac OS, Windows, and Unix).

8.3 New PC Users

A common concern among new PC users was determining just which systems to buy and how to complete meaningful work with their new hardware and software

12. This assessment of gender employs the binary categories prevalent in the early 1990s in America, which is problematic but consistent with the historical and technical sources from the period.

13. For additional insight into this dynamic, see Joy Lisi Rankin, *A People's History of Computing in the United States* (Cambridge, MA: Harvard University Press, 2018), 50.

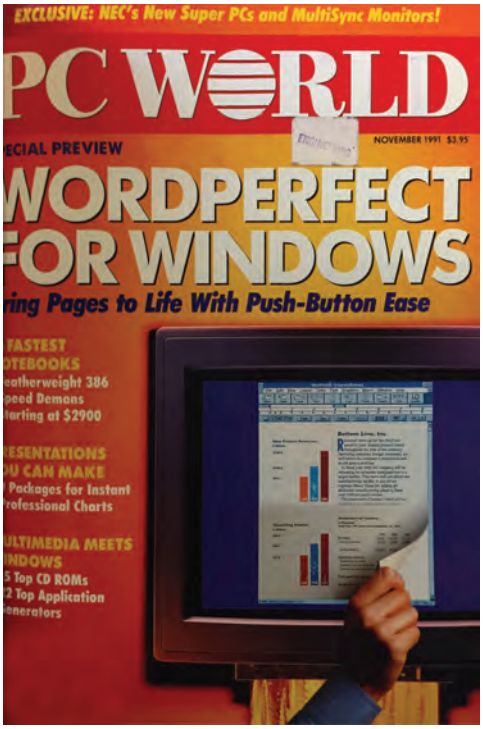


Figure 8.3 ***PC World* magazine (November 1991).** (Used with permission of PCWorld Copyright ©2019. All rights reserved)

platforms. But what happened when companies purchased systems from competing manufacturers? James C. Carlson of Portsmouth, Virginia, wrote to *PC World* magazine in November 1991 to pose this question. (See Figure 8.3.) His team wanted to figure out how to get his company's Macintosh and DOS-based PCs to work together in an office where both hardware and software systems existed side by side. Carlson writes: "My company recently purchased several Macintoshes. All our other computers here are PCs running DOS, and we don't have a network. On the Macs, we run Microsoft Word and Excel. What hardware and software do we need in order to take files from these apps, modify them on our DOS machines, and return them to the Macs?"[14]

Carlson voiced a common frustration about compatibility that was not adequately addressed by Microsoft and other software publishers in the late 1980s and early 1990s. By 1991, Microsoft Word and Microsoft Excel were the top selling software applications for Macintosh systems, even though they were created by

14. James C. Carlson, "The Help screen," *PC World*, November 1991, 39.

Microsoft. These programs were also popular on DOS-based systems running Windows 3.x, but the software was not identical. Of course it seemed like Word or Excel users should be able to open a document on one platform, make changes, save them, and then continue editing the file on another platform—but this was far from a simple task in 1991. First, the document formats and available features were different on the Macintosh and Windows platforms. But there were also complicating factors related to the physical formatting of disks, which differed between IBM PCs and Macs. Although a subtle solution was available—users could format diskettes in a special way and use a format that was compatible with both systems—James Carlson was right to complain. "Why are the systems so hard to use?" His problem gives us an insight into why computer magazines were so popular, and why businesses needed both a clear purchasing strategy and support from an experienced information technology (IT) team.

Another letter in this category addresses the shortcomings of PC platforms head on. Walter Sheehan of Portland, Texas writes to *PC/Computing* magazine about the Windows platform in mid-1991: "Windows is a piece of garbage—an over touted, memory-hungry sham!"[15] Coming to the same conclusion, John R. Egerton of Salida, Colorado writes: "To paraphrase Agent Cooper from 'Twin Peaks,' Windows is an alleged solution to a problem that doesn't exist."[16] The sentiment about Microsoft Windows 3.0 is obvious. But the second media reference may take a little unpacking. *Twin Peaks* was a mystery TV drama created by Mark Frost and David Lynch that debuted on American television in 1990. The series followed the exploits of Special Agent Dale Cooper (played by Kyle MacLachlan) who used quirky methods to solve crimes near the fictional town of Twin Peaks (actually North Bend, Washington). In real life, this small town is about a 45-minute drive from the Redmond campus of Microsoft Corporation. If Windows had a purpose, Egerton pokes, it is hidden away in the minds of its enigmatic creators.

Sheehan and Egerton were not advocates for the Macintosh platform but MS-DOS users who liked their text-based, command-line interface and had little use for Windows 3.0. To many, the Windows product seemed bulky and slow, especially on older hardware. Jeannine M. E. Klein of Houston, Texas shouted from the roof tops in a similar fashion to *PC/Computing*: "I HATE WINDOWS! It takes great heaps of equipment for slow performance, it's crotchety about running anything not specifically (and therefore expensively) designed for it, and most important, it wants to make everybody do everything its way!"[17] Jeannine pulled no punches,

15. Walter Sheehan, "Letters," *PC/Computing*, July 1991, 31.

16. John R. Egerton, "Letters," *PC/Computing*, July 1991, 31.

17. Jeannine M. E. Klein, "Letters," *PC/Computing*, July 1991, 31.

and very succinctly listed the major complaints that users were having with the new graphical user interface (GUI) system.

But were there any Windows 3.0 supporters willing to defend the emerging platform? There must have been some, because by the time of their writing the installed base of Microsoft Windows 3.0 had well-surpassed the Macintosh and OS/2 platforms in terms of users in the U.S. One brave reader, Frank Patton of Moorestown, New Jersey, found that he could defend Windows in print, emphasizing the value of standardization, which might save money for businesses over time. He also brought up a mantra of the technological enthusiasm movement, *productivity*.

> What is Windows doing for us? It's providing a common interface for our applications. Menus and resulting work flow have a continuity that has improved our productivity. We find it easier to train new employees. We can spend more time refining our approaches to problem solving and capitalizing on business opportunities. Windows also uses our system memory more effectively. Our standard PC is a fast 286 with 2MB of RAM, a fast hard drive, and a standard VGA display.[18]

New users were not just interested in the platform wars, which sorted customers initially into opposing campus. They also wrote to computer magazines to get technical support for their systems and to request certain types of information. Mark Thompson complained to *Macworld* in August 1991 that the publication had raised the level of its technical content far too high. In fact, in his opinion *Macworld* was now marginalizing new Mac users (the very people it was supposed to help). Thompson writes:

> You have forgotten your roots. You, like Apple, have left the home user behind. I find almost no articles or product reviews for the home user. Apple is coming around; I hope you also remember those who got you started.[19]

Letters were powerful, but were there other avenues of more immediate product support in this age before the Internet?

Importantly, many new users in the early 1990s still wanted to pick up a telephone and call a company's product support team to ask questions directly about their new hardware or software. This was a privilege that had long been enjoyed by consumers in the U.S. microcomputer industry. Due to the rapid growth of the hardware and software sectors, however, the support infrastructure of many organizations was being overwhelmed. New help desk worker positions were gradually

18. Frank Patton, "Letters," *PC/Computing*, July 1991, 31.

19. Mark Thompson, "Letters," *Macworld*, August 1991, 35.

created to handle the avalanche of requests, which is a development that deserves further study by historians of labor and technology.[20] The long-term consequence of this call volume would be to push customers to Internet-based tools to resolve their issues, and to establish lower-cost call centers in locations like India to save money. But in 1991, most of these support functions were still being handled by regional sales offices and in-house engineering teams, just as they had been during the early days of PCs. Letters to the editor allow us to eavesdrop on the impact of these evolving support and labor issues. For example, in September 1991, David Sprogis of Belmont, Massachusetts, wrote to *Macworld* about his Epson printer purchase, complaining:

> Who was the Einstein at Epson that put technical support on a 900 line? The cost of Epson's technical support is $2 per minute, excluding the first minute, which is free. I don't want to pay to get a few technical answers concerning the purchase of a new item. Paying for the printer should be the bottom line.[21]

So much for passing the cost of customer service back to the customer. But product support wasn't always this bad, as Ted Neff from Nichols, Iowa shared in a letter in the October 1991 issue of *PC World*:

> I want to express my appreciation to Borland for their service when I upgraded to Quattro Pro 3.0. While installing my upgrade, I discovered that the fourth disk in the set was damaged. After I contacted Borland through their CompuServe forum, they shipped me a complete set of new disks at no charge via Federal Express—without my requesting express delivery.[22]

As this letter reminds us, all software fixes had to be sent out through U.S. mail or via overnight service (Federal Express), which typically impressed customers and demonstrated a company's commitment to its users. Companies were also exploring alternatives to support requests that arrived via phone calls and letters. As a forerunner to commercial web browsers, CompuServe Information Service (CIS) provided dial-up access to user-run *support forums* for computer users, which provided technical information as well as news about sports, politics, entertainment, and popular hobbies. CIS customers made a connection to CompuServe's proprietary servers using a modem and a telephone line, then they paid an hourly fee to

20. One of the few scholars to introduce the issue is Greg Downey, who also studied Computer Science and worked in the computer industry in the 1980s and 1990s. See Greg Downey, "Virtual webs, physical technologies, and hidden workers: The spaces of labor in information Internetworks," *Technology and Culture* 42, no. 2 (April 2001): 209–235, here at 230.

21. David Sprogis, "Letters," *Macworld*, September 1991, 58.

22. Ted Neff, "Consumer watch", *PC World*, October 1991, 58.

use the service. CompuServe started offering online experiences in the 1980s, and they rapidly expanded them in 1991, when the company claimed that it possessed the world's largest network of people with PCs. In a *Macworld* advertisement published the same year, CompuServe boasted that "brains abound on CompuServe." Product support could be unpredictable elsewhere, they noted, but "we have the world's largest network of people with answers to your hardware and software questions."[23]

This type of competition gradually pressured hardware and software manufacturers to improve their technical support, although many soon transferred this work to overseas call centers. In the November 1991 issue of *Macworld*, director Dave Christopherson of Epson America wrote to reassure his company's frustrated customers, "In response to the letter in your September 1991 issue about Epson's 900 technical support line, we are pleased to say that on July 23 we reinstated our toll-free consumer-support number: 800/922-8911."[24]

In short, the customer revolt articulated on the pages of *Macworld* succeeded.

8.4 Power Users

Power users expressed their concerns in different ways in computer magazines. Often, they found it important to project an aura of expertise in their letters, rejecting the claims of industry pundits or showing off their technical skills. For example, in a Fall 1991 issue of *PC Magazine*, Michael R. Kabala of Eldridge, Iowa wrote to quibble with John C. Dvorak, a prominent *PC Magazine* columnist. Dvorak had lamented that the era of the DIY "hobbyist" was over in personal computing, a claim that Kabala disputed. Both men made reference to the legendary Home Brew Computer Club, the iconic San Francisco Bay Area organization that held mythical status for PC power users and programmers. (For more on Home Brew, see Chapter 2.) Kabala writes:

> I must take issue with John C. Dvorak's "Whither the Hobbyist?" column (June 11, 1991). While not many people these days are putting together their own PCs using a pile of chips and a soldering iron, the "home-brew" computer is still very much alive, but in a slightly different form. The basic building blocks are now motherboards, power supplies, and disk drives. True, this amounts more to assembling than building, but it still helps to have some of those hacker skills. Don't pronounce us dead yet—I can still feel a pulse.[25]

23. CompuServe [paid advertisement], *Macworld*, August 1991, 97.

24. Dave Christopherson, "Letters," *Macworld*, November 1991, 42.

25. Michael R. Kabala, "Letters," *PC Magazine*, September 24, 1991, 18.

In the same issue of *PC Magazine*, an IBM PC "clone" user wrote to the "Advisor" column seeking help with inserting memory into an aging computer system. Nicholas Tea of Long Island City, New York, felt that he could install the memory chips in his 386-based computer, but he was concerned about the price, and presumably looking for advice about a mail-order vendor that might offer a good deal. Tea's letter emphasizes his power user expertise, and also some of his tinkering abilities.

> I would like to increase the amount of memory in my CompuAdd 386 PC. It came with 1MB of 256K chips and I would like to replace them with 4MB of 1MB SIMMs. I had read in a past "Pipeline" column that 1MB SIMMS have dropped in price. I called up CompuAdd and they charge almost $100 per megabyte. It seems like others are charging from $50 to $60. Are they proprietary? Can I purchase the cheaper SIMMs from somewhere else and mount them myself? If so, where?[26]

Colleen O'Hara, a power user from Terre Haute, Indiana, wrote in a similar way to *PC World* in August 1991, hoping to get some help managing a complex system-level task. She noted that "memory-resident" applications were causing trouble with Windows 3.0 and also her computer aided design (CAD) program. Memory resident utilities were often called TSRs or "terminate and stay resident" applications. Under certain circumstances, the user could close a program (removing it from the screen) but it would remain in memory, hidden from view. This is a common feature of smart phones and multitasking operating systems today, but in the early 1990s the phenomenon was relatively new and poorly understood in PC communities. O'Hara writes:

> Is there a way I can use different drivers and TSRs (memory-resident utilities) with the DOS applications I'm running under Windows 3.0? When running my CAD program under Windows, I'd like to be able to pop up PC Tools' memory-resident DOS shell and calculator and also be able to use the third mouse button (which requires a special driver), but I can't get this setup to work properly.[27]

Computer journalists were aware that PC power users wore many hats, and they had little time to acquire programming skills, despite their importance in the

26. Nicholas Tea, "Advisor," *PC Magazine*, September 24, 1991, 459.

27. Colleen O'Hara, "The Help screen," *PC World*, August 1991, 41.

workplace. This tension is sometimes visible in essays and advice columns from the era, such as this excerpt from an article on databases, written by Richard Scoville in the July 1991 *PC World*.

> If you must keep track of people, purchases, inventory, or other information, you need a data manager. You probably already know that, but you're reluctant to buy one—who can spare two years in a monastery learning dBASE programming?[28]

Ashton-Tate released PC versions of dBASE for CP/M, the Apple II, and IBM PCs and compatibles in the 1980s. In 1991, Borland International acquired Ashton-Tate and its product line, hoping to benefit from the estimated three million dBASE users in the marketplace.[29] dBASE programming involved writing procedural commands to open and move through database records in one or more files. Numerous books were written for dBASE programmers to teach them the fundamentals of programming, but it was considered a complex subject by most in the business community.

Power users *were* DOS batch file programmers, however. In the August 1991 edition of *PC/Computing*, columnist Jean Atelsek wrote a spirited call-to-action for DOS batch file enthusiasts, arguing for the importance of batch programming as a time-saving activity. (See Figure 8.4.) Her article began with a tongue-in-cheek statement about people's addiction to PCs:

> PCs are dumb! Chances are you couldn't do your job without one. But since the day you first put your fingers on the keyboard, you've been telling that stupid machine what to do. Gives you a sense of power, doesn't it?
> But issuing DOS commands isn't enough: You have to write batch files and DEBUG scripts to make your PC do your bidding instantly. Sure, spreadsheets and word processors are handy, but why bother with those keystrokes if there's a faster, smarter way?[30]

In the presentation that followed, Atelsek presented 25 pages of DOS batch files sent in from *PC/Computing* readers. Each project was accompanied by a rationale for use, a code listing, and operating instructions. I reproduce the listing

28. Richard Scoville, "Data Management: Up close and personal," *PC World*, July 1991, 142.

29. Carla Lazzareschi, "Borland to acquire Ashton-Tate in a $439-million deal," *Los Angeles Times*, July 11, 1991.

30. Jean Atelsek, "Working smarter," *PC/Computing*, August 1991, 96–121, here at 96.

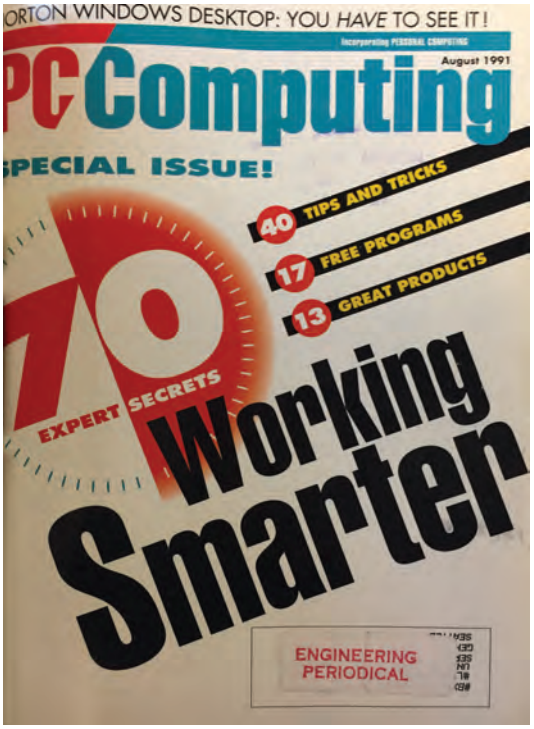


Figure 8.4 ***PC/Computing* (August 1991).** (Reprinted with permission. ©2019 Ziff Davis, LLC. All Rights Reserved)

for Move.bat below, along with operating instructions, as a sample of the routines that were sent in.[31] Move.bat is sometimes referred to as "the command that DOS forgot," because the utility quickly moves a file from one folder to another. Power users were so enamored with this batch file that the MS-DOS development team eventually added a Move command with similar functionality to MS-DOS version 6.0.

```
@ECHO OFF
IF %2. == . GOTO INCOMPLETE
XCOPY %1 %2
IF ERRORLEVEL 1 GOTO ERROR
DEL %1
GOTO END
:INCOMPLETE
ECHO You must specify the destination path.
:ERROR
```

31. Jean Atelsek, "Working smarter," *PC/Computing*, August 1991, 96–97.

```
ECHO Something's wrong. Aborting move.
:END

Be sure to put MOVE.BAT in the same directory as XCOPY.EXE
(probably the DOS directory), and make sure that directory is
in your path. To use MOVE.BAT, simply type MOVE followed by the
name and the path of the file you want to move, a space, and
the destination directory.
```

The parameters %1 and %2 represent the file to be moved and the location folder, respectively. If the second parameter (%2) is not specified on the command line, a GOTO statement branches to the "INCOMPLETE" label, and the batch file prints the message "You must specify the destination path." If another error takes place (if the filename has been misspelled or the wrong number of parameters is specified), a GOTO statement branches to the "ERROR" label, and the message "Something's wrong. Aborting move." is printed. However, if the filename and location are recognized, the DOS XCOPY command copies the file to a new location, and the DEL command deletes the original file.

8.5 Advanced Hobbyists

In the early 1990s, the identity of advanced hobbyists was shifting from power user to programmer as new opportunities presented themselves to upwardly mobile computer users. Advanced hobbyists often had deeper proficiencies with hardware and software platforms, and in many cases they had used earlier systems, such as IBM mainframes or the DEC PDP-11. Gurdon Abell of Woodstock, Connecticut wrote to *Dr. Dobb's Journal* in January 1991 about his past experiences, framing himself as a computer polymath with an awareness of programming that took him up to the boundaries of professional software development. He calls himself an "electronischer," and his letter is a fascinating artifact from the age of digital self-fashioning, *c.* 1991.

> An Accidental Tourist...
> Dear DDJ,
> I am not a programmer, amateur or professional, but a semi-retired physicist, electronischer [sic], inventor, and patent buff who is much more at ease bashing tin or slinging solder than writing the strictly ordered poetry of a computer program. Oh, all right, I do cook up some Basic, with a few lines of machine code thrown in, to do donkey-work (modeling the on-axis performance of any horizontal-axis windmill using a Sinclair ZX-81 and only 16K; or home-brew memory-mapped I/O, using another Sinclair to control and log a long-term

> life test) which I would otherwise have to do manually. To me, a computer is just another power tool, like a sabre saw.
> So why on earth am I a subscriber to DDJ? It was an accident… It proved to be highly technical, but in a field which was (and is) very strange and wonderful. Your accidental subscriber has found *Dr. Dobb's*.
> But why have I continued to subscribe? … Even though I am nearly illiterate in the languages of programming, I can enjoy exploration of neural nets, fulminations about Ada, and always the trenchant comments in "Swain's Flames." My interest seems more aesthetic than technical, but that may be a common human trait: I thoroughly enjoy opera, even though I know little German and French, and even less Italian.[32]

Among the various references to 1980s and 1990s computing culture in this letter are programming a Sinclair laptop using BASIC and reading the columns of Michael Swaine, the co-author of the popular computing history *Fire in the Valley*. Gurdon Abell is showing off some but also identifying a location for himself in the milieu of hobbyist computing and PC culture.

This technical world was rapidly changing, as software consultant Jim Westrick from Washington, DC, noted in his April 1991 letter to *UNIX World*. This magazine enjoyed support from a wide cross-section of Unix users, including those on microcomputer platforms. Westrick instructed others how to write hobbyist-style programs, and he had recently switched from C to AWK, a system-level utility designed for data extraction. His letter encourages other Unix advocates to try the same approach.

> Dear Editor:
> As UNIX moves into the commercial marketplace, we are seeing new types of users and programmers. They are not as astute as the first UNIX users, perhaps not even capable of writing the C "Hello World" program. In fact, if it were not for Foxbase, Informix, or other database systems, they would not be using the UNIX multiuser platform at all.
> They may have their databases, but most often the program [that] they use to convert their old files, manage odd jobs, and use as their general utility program is awk… This simple eloquent programming language is the perfect utility for modern databases on most commercial applications…. As a consultant, I do not try to teach new customers C. I give programmers a basic course in awk…"

32. Gurdon Abell, "Letters," *Dr. Dobb's Journal*, January 1991, 14.

Jim Westrick
Senior Consultant
KPMG Peat Marwick[33]

But were advanced hobbyists really interested in the Unix operating system? Good PC-based versions of Unix were certainly available in the early 1990s, including Unix System V from Santa Cruz Operation (SCO). Microsoft sold versions of Xenix and Unix in the mid- to late 1980s, but gradually neglected Unix in favor of its own versions of MS-DOS, Windows, and OS/2. An exchange between *Byte* columnist Jerry Pournelle and an advocate for Unix reveals some of these tensions—a second front that was somewhat less visible during the "platform wars" of the era.

Pournelle began the debate with a 1991 *Byte* column describing how he had installed Unix on his 80386-based PC and experimented with it. After a few weeks, he uninstalled the system in disgust because it was too cumbersome to operate. William B. Fankboner of La Quinta, California took umbrage with Pournelle, writing a letter that described Pournelle as impatient and "prematurely geriatric." Tools like Unix and C took a lifetime to master, Fankboner acknowledged, but the rewards were worth it. Fankboner concluded:

> Jerry Pournelle seems to be encouraging a certain impatience with recalcitrant software. His attitude seems to be that if a program doesn't do his instant bidding, he'll stuff it into the nearest waste can... Life is too short to spend a lot of time thrashing around helplessly with some perverse command structure.
>
> On the other hand, approaching software with a chip on your shoulder can be distinctly counterproductive... Jerry's battles with Unix and C are premier examples. He dutifully installed Unix on one of his 386s, and when he couldn't get it to run satisfactorily in two weeks, he consigned it to limbo because it was "unfriendly."
>
> Neither Unix nor C is for the faint of heart or the prematurely geriatric. This is software that requires a lifetime commitment. Expect to work hard for several months before you gain even minimal control over a system and a language as powerful as Unix. Once you have made this initial investment of energy, the rewards are big-time.[34]

Pournelle was recalcitrant, however, and used to regular criticism as a columnist.

33. Jim Westrick, "Mail," *UNIX World*, April 1991, 26.

34. William B. Fankboner, "Letters," *Byte*, August 1991, 20.

8.6 Professional Programmers

Professional programmers were also active participants on the PC platform. They rarely postured or complained about buggy software, but wrote instead to offer solutions to challenging problems or to express ideas that they were passionate about. A typical example is this regard is an August 1991 letter from Rodney Hills of Gresham, Oregon, published in *Byte* magazine. Hills wrote to describe his interest in a recent article that he had read in *Byte* about sorting algorithms. Hills explains that he implemented an algorithm in a FORTRAN program, then found a way to improve it, and share it with readers.

> The April article "A Fast, Easy Sort" intrigued me so much that I had to try it out. I coded it in FORTRAN 77, ran it, and was well pleased with the performance. There was one line of code in the program that I took exception to:
>
> ```
> gap=(int) (float)gap/1.3);
> ```
>
> This equation requires that the integer gap be converted to floating point and divided by 1.3 in floating point, and the result converted back to integer. In removing the floating-point operations, I was able to see a 15 percent improvement in CPU performance.[35]

This type of community-based interaction was important for the members of Code Nation, and it took place regularly in computer magazines and academic journals. Hills was either a professional developer or a self-taught coder who was working to develop commercial or scientific skills.

Harry Smith, another experienced developer from Mountain View, California, wrote to *Dr. Dobb's Journal* in June 1991 about a problem that he was having with math coprocessors. He expressed frustration with an incompatibility that he had noted between systems, an incompatibility that Smith discovered between the Intel 80287 chip and various IBM PC AT "clone" computers using 80386/80387 chips. Smith discovered the problem when he tested code from a popular book on graphics programming entitled *Fractal Programming in C*, by Roger T. Stevens. Smith then outlined how fixed the problem for the benefit of readers.[36]

> Dear DDJ,
> I have a problem: A certain floating point multiply instruction does not work correctly on my 80386/80387-based AT clone machine. This has been tested on 387s in machines of four different manufacturers and they have all failed.

35. Rodney Hills, "Letters," *Byte*, August 1991, 18.

36. Roger T. Stevens, *Fractal Programming in C* (Redwood City, CA: M&T Books, 1989).

> It has also been tested on several 287s and 8087s and they have all worked correctly.
> The problem instruction was first found after compiling the program CNEWTON3.C from the book *Fractal Programming in C*, by Roger T. Stevens, using Borland's Turbo C 2.0. The program's screen output in certain regions was a solid brown color when it should have been varying shades of blue. By using the debugging aids of Turbo C, the problem was traced to a double-precision floating point multiply instruction compiled from lines of C code…
> Harry J. Smith
> Mountain View, California[37]

What role did the ACM's flagship magazine, *Communications of the ACM*, have for professional programmers on the PC platform? In January 1991, the readers and editors of *Communications of the ACM* were engaged in a lively discussion about how the publication might strike a balance between academic articles and content for working programmers in industry. As *Communications of the ACM* was a magazine for the *entire* membership, it seemed important to adjust its format periodically, which several letter writers noted and appreciated. However, the mythology that computer scientists were occupied primarily with theoretical problems remained a stumbling block for some. ACM Member John Buffum from Leavenworth, Kansas, shared his thoughts on the issue in "ACM Forum":

> I have been working on computers now since 1983. I hold a Master's Degree with thesis. I am no stranger to computers and research. I am no dummy.
> I am, however, having a rough time with your magazine. As I have a 100-year old house, am a student at a local college, and have a full-time job and a family, time is precious. Currently, I subscribe to *Dr. Dobb's Journal* and *Computer Language*. Though they do not "push the envelope" the way your magazine does, they are very easy to read and contain excellent and useful information. A DDJ or CL article can be read in 15 minutes with good digestion. A *Communications* article, with precious few exceptions, requires one to two hours and sometimes outside help…
> What I ask is re-examination of the line you draw between scholarly and unnecessarily convoluted… Some of the terms you use and some of the information you assume we know, we do not…[38]

37. Harry J. Smith, "Letters," *Dr. Dobb's Journal*, June 1991, 14.

38. John Buffum, "ACM forum," *Communications of the ACM* 34, no. 1 (January 1991): 16–17.

IEEE's *Computer* magazine was a venerable engineering periodical that could also feel out of step with the PC software industry. In September 1991, the 40-year-old publication (at that time) celebrated its past and looked to the future, publishing a feature article on emerging GUIs. Authors Aaron Marcus and Andries van Dam entitled their essay, "User-Interface Developments for the Nineties."[39] However, in the entire article on GUIs there was no mention of Microsoft Windows, OS/2, or Mac System 7. In the PC marketplace, there was only a passing mention of Xerox and Apple as "popularizers" of the GUI in the past. The authors did show a screen shot of Open Software Foundation's Motif user interface, but the article limited its discussion of Motif to rudimentary elements such as the Menu bar, Title bar, Scrollbar, and a few buttons. Compared to PC industry publications like *Byte*, *Dr. Dobb's Journal*, *PC Magazine*, and *Macworld*, the article seemed out of touch with contemporary experience, where programmers were at work learning complex GUI tools, skills, and software development kits (SDKs). Understandably, the disconnect may have been because the article's peer review process for *Computer* took longer than trade publications. But no additional articles on GUIs would appear in *Computer* for the remainder of the year. The division between the academy and working professionals was at best a persistent mythology, but it lingered on due to omissions like this.

We conclude our review of user responses to computing culture with a submission from Stephen Bobic, a self-professed hacker and programmer from Oak Ridge, Tennessee. Bobic wrote a letter published in the June 1991 *Macworld* magazine asking for a little more respect for the hacker tribe, who Bobic argued could improve the robustness of computer systems and help system administrators locate security breaches. Bobic seems to be reflecting the positive view of hacking that emerged after Steven Levy's *Hackers: Heroes of the Computer Revolution* (1984). Levy described hackers as brilliant trailblazers who took social and personal risks with computers but ultimately moved programming and computing forward. Stephen Bobic writes:

> As a hacker and a programmer, I am completely appalled at the lack of respect that computer programmers, users, and hackers get. Face it, folks, hackers are actually a good thing. We make your system administrators aware of security holes and make them do their jobs. We hackers are an inconvenience; we are not a true threat. For our government to allow this kind of fifties-mentality ransacking of a business is unconscionable.[40]

39. Aaron Marcus and Andries van Dam, "User-interface developments for the nineties," *Computer*, September 1991, 49–57.

40. Stephen Bobic, "Letters," *Macworld*, June 1991, 36.

The final sentence makes reference to a recent arrest of prominent hackers, which had been circulated in the news media and prompted soul searching among computer professionals. Bobic's reference to a "fifties-mentality" and government agents involved with the "ransacking of a business" evokes images of the Cold War and McCarthyism, periods of surveillance that most baby boomers wanted to forget.

As a barometer of the debate, the ACM performed its usual role of exploring the issue from multiple perspectives and explaining how hacking and security violations impacted the computer industry from the editors' point of view. In the March 1991 issue of *Communications of the ACM*, the editors published a multi-part article on several cases involving electronic publishing, constitutional rights, and computer programmers who were accused of hacking.[41] The coverage explored in detail the 1990 U.S. District Court case involving Craig Neidorf, a U.S. college student accused by the U.S. government of fraud and interstate transportation of stolen property. Neidorf was prosecuted because of a document that he published in the electronic newsletter, *Phrack*. Coverage of the case in *Communications of the ACM* was balanced, but there was clearly a rising level of concern about what hackers had done and what they might do in the future. Considering all the issues, what should the government do about online trespassers who illegally entered systems and redistributed sensitive information?

Gordon Meyer's comment in the essay's commentary section seemed to sum up the debate and take critiques of hacking in a new direction. Meyer recognized the antisocial nature of contemporary hacker-intruders, but he also argued that the threat was minimal, or at least not as sinister as many believed:

> The computer underground is a marginally deviant subculture. It's not as sophisticated, not as conspiratorial as once thought; and it's not full of antisocial sociopaths as once described.[42]

In historical perspective, Meyer was probably right to point out that the goal of hackers was not to encourage crime or antisocial behavior through programming. Rather, the activities of hackers—as well as power users, gurus, tinkers, and other coders working on the margins of society—was oriented around making existing systems work better, not worse. Although these groups sometimes had objectives and worldviews far removed from the aims of polite computing society, both groups were essentially part of the same project.

41. Dorothy E. Denning, "The United States vs. Craig Neidorf: A debate on electronic publishing, constitutional rights and hacking." *Communications of the ACM* 34, no. 3 (March 1991): 24–43.

42. Gordon Meyer [co-editor of *Computer Underground Digest*], quoted in "Colleagues debate Denning's comments," *Communications of the ACM*, Vol. 34, no. 3 (March 1991): 37.

8.7 New Approaches to Historical Research

As a general conclusion to this presentation of reader responses from the pages of *Byte*, *Communications of the ACM, Dr. Dobb's Journal*, *IEEE Computer*, *Macworld*, *PC/Computing*, *PC Magazine*, and *UNIX World*, it is obvious that PC users were highly aware of efforts by computer companies to establish brands and aggressively market their products. Apple, Borland, Epson, Microsoft, Unix systems, and various "clone" computer manufacturers are all featured on these pages. Moreover, there was significant resistance to the idea that one platform (in this case, Microsoft Windows) should be a comprehensive computing standard that every user should accept. A significant amount of "flame mail" confirms this issue, which conspicuously divided PC users into different factions or "camps." But beyond the well-publicized operating system wars, the letters reveal American consumers who are frustrated with poor quality products, incompatibility issues, support policies, and the costs necessary to buy and maintain PCs. Users are also highly aware of the influential roles that industry pundits and columnists play in presenting information about PCs and computing. Columnists and magazine publishers had a major influence on how new systems were introduced and used by the general public. All of these perspectives are important to consider as we assess how regular people accepted, accommodated, and rejected the machinations of industry elites during the formative years of personal computing.

In terms of programming culture, it is also clear that hobbyists, self-taught programmers, professional developers, and hackers on the margins were reading the same books and periodicals. They participated collectively in a programming culture that was taking distinctive shape around personal computing in the 1970s, 1980s, and 1990s. This observation supports an important argument of *Code Nation*—that in social, economic, and intellectual terms, there is discernable common ground among programmers and the intensive users of computers. This includes shared mental abstractions related to computer literacy, problem solving, consumer culture, print networks, operating system platforms, and other historical contexts. Computer books and magazines are important mediators among these realms, and they are also vital locations for new historical research.

In the next section of this book, Part III: Professional Programming Cultures, I will shift the focus of *Code Nation* from educational movements and nonprofessional programming environments to more commercial contexts of the learn-to-program movement. I will discuss commercial application development for the MS-DOS, Apple Macintosh, and Microsoft Windows platforms, and the business practices that led to influential product evangelism initiatives and lively

commercial computer trade shows. Chapter 9 begins this presentation with an influential group of entrepreneurs, authors, and programmers who created innovative resources for MS-DOS application development, including Peter Norton, Philippe Kahn, Anders Hejlsberg, Ray Duncan, and JoAnne Woodcock.

PART II

PROFESSIONAL PROGRAMMING CULTURES

9 Developing for MS-DOS: Authors and Entrepreneurs

"Obviously, you aren't going to be able to do any assembly language coding for the IBM/PC if you can't handle assembly language. However, there seem to be quite a few people who have done assembly language programming for other computers..."
Peter Norton, *Inside the IBM PC* (1983)[1]

"Given adequate information about the hardware, any competent assembly-language programmer can expect to successfully interface even the most bizarre device to MS-DOS without altering the operating system in the slightest..."
Ray Duncan, *Advanced MS-DOS* (1986)[2]

In Part III, *Professional Programming Cultures*, I shift the focus of *Code Nation* from hobbyist, hacker, and non-professional coding environments to programming communities that hoped to develop commercial applications and operating system utilities for the MS-DOS, OS/2, Apple Macintosh, and Microsoft Windows platforms. This development represents a new phase of the learn-to-program movement, because the goal of aspiring commercial developers was not just to learn a programming language, tinker with communication systems, or improve one's office productivity by writing batch files. Rather, personal computing represented a new commercial platform for aspiring coders, including the prospect of selling mail order games or utilities that might benefit from the surging interest about computers in America. To promote this opportunity, software publishers, computer book authors, and magazine publishers created new learning materials that encouraged

1. Peter Norton, *Inside the IBM PC: Access to Advanced Features and Programming* (Bowie, MD: Robert J. Brady Co., 1983), 238.

2. Ray Duncan, *Advanced MS-DOS: The Microsoft Guide for Assembly Language and C Programmers* (Redmond, WA: Microsoft Press, 1986), 222.

relatively inexperienced programmers to learn software development techniques, including the hidden features of personal computer (PC) architecture, MS-DOS services, device drivers, graphical user interface (GUI) applications, and much more. This marked a new, commercial phase of the learn-to-program movement in personal computing that involved tens of thousands of software developers and set the stage for business and corporate computing culture in the 1990s and beyond.

Rather than a technical manual full of information about commercial programming tools and system services, this chapter explores the technical history of this era through the primers, guidebooks, and reference manuals that aspiring programmers used to create software for the MS-DOS platform. I'll introduce the authors and entrepreneurs Peter Norton, Larry Joel Goldstein, Philippe Kahn, Anders Hejlsberg, and Ray Duncan, each of whom transferred effective software development skills from the mainframe or minicomputer worlds to the software and hardware platforms driven by IBM PCs and compatibles. We'll also examine the technical writing of JoAnne Woodcock, a prolific author and editor who chronicled the early history of MS-DOS and taught advanced users to operate and exploit the features of the MS-DOS operating system.

How important was this relatively hidden world of self-taught MS-DOS programmers and aspiring professional developers? Although commercial MS-DOS development began as a remote backwater in the wider world of professional programming, it grew into a highly popular endeavor by the mid-1980s. By this time, reliable assembly language, Pascal, and C compilers hit the market that could exploit operating system services and create fast, optimized applications. A measure of the cultural impact of this new industry is the commercial success of Peter Norton's *Programmer's Guide to the IBM PC*, a bestselling reference book for software developers that sold over 500,000 copies in its first two editions (1985 and 1988).[3] Ray Duncan's "Power Programming" column in *PC Magazine* also became a leading source of information in the PC programming world by the mid-1980s. His well-written technical articles created a sense of possibility among new-to-topic programmers who were looking to strike it rich in the emerging world of commercial DOS development. The success of this work led to the publication of *Advanced MS-DOS* (1986) and *The MS-DOS Encyclopedia* (1988), two of the era's definitive DOS programming references for commercial developers.[4]

3. Peter Norton, *The Peter Norton Programmer's Guide to the IBM PC* (Bellevue, WA: Microsoft Press, 1985); Peter Norton and Richard Wilton, *The New Peter Norton Programmer's Guide to the IBM PC and PS/2*, Second Edition (Redmond, WA: Microsoft Press, 1988).

4. Duncan, *Advanced MS-DOS: The Microsoft Guide for Assembly Language and C Programmers*; Ray Duncan, ed., *The MS-DOS Encyclopedia: Versions 1.0 to 3.2* (Redmond, WA: Microsoft Press, 1988).

Why were seemingly obscure programming texts like this in such high demand? First, because the installed base of IBM PCs and compatibles rapidly expanded in the mid-1980s, encouraging lucrative financial dreams for the entrepreneurs who could create and market applications for the emerging platform. Although the underlying hardware and software systems for DOS-based PCs were relatively limited, the promise of coding "secrets" from programming authors could transform obscure memory allocation routines, function calls, and interrupt handlers into software gold. In fact, the IBM PC/MS-DOS platform would only survive and flourish if third-party advocates dispersed programming techniques to the legions of self-taught programmers who were struggling with systems that were buggy, poorly documented, and evolving quickly. A truism in the commercial electronics industry was that well-designed platforms drove innovation and technology diffusion.[5] Beyond price structure, the major reason for a new platform's success is that the adherents of an emerging standard taught others how to use and profit from the new system, persuading them that it would be beneficial to convert and make further investments in time and money.

This chapter and Chapter 10 will survey many of the popular programming tools and resources that helped PC programmers create applications for the MS-DOS, OS/2, and Windows platforms from 1982 to 1993. My goal is to explore how advanced programming techniques were diffused by computer books, magazines, and software systems, with an emphasis on the authors of operating system references. Collectively, these tools helped move the learn-to-program movement into the realms of commercial programming and professional networks. Chapter 11 will build on these ideas by examining the rise of enterprise-computing platforms, and the systems that professional developers used to support corporate and client-server computing. I'll examine the commercial marketplace of programming tools, industry trade shows, certification programs, and product evangelism—all "professionalizing" aspects of the PC industry, with important consequences for the learn-to-program movement.

9.1 New Platforms for Commercial Software

As personal computing gained momentum in the late 1970s, numerous PC software firms were established that collectively sold thousands of applications for the new platforms. This process began with the release of pioneering programs and games for the Apple II, Tandy TRS-80, and Commodore PET microcomputers, and it gained momentum through the introduction of IBM PCs and compatibles in the early

5. David S. Evans, Andrei Hagiu, and Richard Schmalensee, *Invisible Engines: How Software Platforms Drive Innovation and Transform Industries* (Cambridge, MA: The MIT Press, 2006), 109.

1980s. By the end of 1983, there were some 3,000 venders producing an estimated 35,000 PC software products in the commercial marketplace, an increase of 50% from the previous year.[6] The earliest products included operating systems and programming languages from companies such as Digital Research and Microsoft. These tools allowed users to experiment with PCs and write their own programs in BASIC and other languages, although the capabilities were limited. Soon the rudimentary tools were supplemented by new versions that were built for computers with more capabilities. These programs included electronic spreadsheets, word processors, databases, graphics programs, communications tools, productivity utilities, educational software, payroll and accounting packages, and a selection of specialized applications for home and business use. Figure 9.1 shows how one corporation, IBM, tried to market their suite of applications in retail contexts.

Video games for IBM PCs and compatibles became one of the most important categories of commercial software. PC games grew from a modest collection of "adventure" or "breakout" style programs into a robust genre that would come to dominate other forms of electronic media. The first IBM PC games included best-sellers such as *King's Quest* (Sierra On-Line), *Zork* (Infocom), and *Microsoft Flight Simulator* (Microsoft). Although the 1983 video game "crash" demonstrated how tenuous this new market could be, entertainment software continued to be a major area of interest for consumers. Video gamers who owned a PC and a BASIC interpreter could also experiment with writing or editing their own games via widely-circulating source code listings. (See Chapters 5 and 10 for more information about video game programming resources.) Many computers, such as the IBM PC XT, included a version of classic BASIC in read-only memory (ROM), making it easier to load and run games written in the language. There were also numerous computer magazines that specialized in video games, and most of these published program listings to feed users' interests in game development. These magazines included *ANALOG Computing*, *Computer and Video Games* (UK), *Computer Gaming World*, *Computer Fun*, *Electronic Games*, and *Videogaming Illustrated*. New media scholars and historians are now at work documenting this fascinating world.[7]

What difficulties did early PC software makers encounter? The real challenge for commercial developers was that the emerging platforms were new and there

6. Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: The MIT Press, 2003), 208.

7. See Henry Lowood and Raiford Guins, eds., *Debugging Game History: A Critical Lexicon* (Cambridge, MA: The MIT Press, 2016); Jimmy Maher, *The Future Was Here: The Commodore Amiga* (Cambridge, The MIT Press, 2012); Nick Montfort and Ian Bogost, eds., *Racing the Beam: The Atari Video Computer System* (Cambridge, MA: The MIT Press, 2009); and Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog*, chapter 9: "Home and recreational software."

Figure 9.1 **Kiosk for purchasing IBM PC software applications and programming tools, *c.* 1984. The software boxes visible include IBM Logo, IBM Writing Assistant, and IBM DisplayWrite 2. IBM put considerable effort into developing its own line of PC software products after the release of the IBM PC XT.** (Courtesy of the Computer History Museum)

were few established tools and compilers to build applications. Although some of the first PC programmers did have industry experience with mainframe and minicomputer systems, the languages and compilers were different on PCs, and so were the learning systems and management structures within corporations. For example, many PC software companies were little more than basement start-ups run by energetic entrepreneurs who were learning how to manage people and create commercial software at the same time. To exacerbate these problems, the resources on PCs were severely limited, with system memory, disk space, input/output, and software services all constraining factors. The software developers who figured out the tricks to make PC hardware and software systems perform well could contribute greatly to a new company's bottom line. These entrepreneurs could also supplement their income by teaching others how to write programs, distributing their ideas via computer books, magazines, and newsletters devoted to the new platforms. Newcomers to programming could then follow in their footsteps.

9.2 *Inside the IBM PC* with Peter Norton

One such programmer/entrepreneur was Peter Norton (1943–), who grew up in the Seattle area and attended Reed College near Portland, Oregon. Norton attended Reed a decade *before* Steve Jobs did, graduating in 1965 with a Bachelor's degree in Mathematics and Philosophy. Both men were powerfully influenced by their experiences at the small liberal arts college. Norton developed a life-long appreciation for art, philosophy, and the importance of a liberal arts curriculum. Years later, when Reed formally established Computer Science as a concentration within their Mathematics program, Reed officials asked for Norton's advice. The writer and software entrepreneur stressed the importance of style and aesthetics as a foundation for technical training. "Elegant coding is faster and more reliable," Norton wrote. "We know that aesthetics matter in interface design. So let's stir art, art history, and graphic design into our computer science major."[8] In appreciation for his ongoing influence and support, Reed College made Peter Norton a trustee.

After graduation, Norton traveled some and then spent several years in a Buddhist monastery in the San Francisco Bay Area.[9] He was also employed by The Boeing Company and the Jet Propulsion Laboratory (JPL), two successful aerospace companies that were benefiting at that time from generous government contracts. Throughout the 1970s, Norton had regular exposure to mainframe and minicomputers, learning the ropes of software development and gaining experience with system utilities that made regular use of system memory and other hardware components. When the cyclical downturns of the aerospace industry left Norton without a job, he looked for something different and turned his attention to the new IBM Personal Computer, which had just been released in August of 1981. Norton purchased one of the first IBM units and began experimenting with it, learning MS-DOS and writing simple programs in BASIC and assembly language. In just a few months, Norton had more experience than most PC tinkerers and users on the new platform, and he was able to adapt his formative coding experiences with earlier systems to the smaller world of the IBM PC. He became fascinated with the internals of the PC—the computer's ROM-BIOS services, command processor (COMMAND.COM), and the layout of early floppy disks. (At the time, this removable storage media was 5.25" wide and could store 160KB of information on its single-sided format.) The first IBM PC did not come with a hard disk drive, but it shipped

8. Peter Norton quoted in, "Digital pioneers help Reed design CS program," *Reed Magazine* 94, no. 3 (September 2015). https://www.reed.edu/reed_magazine/september2015/articles/eliot_circular/cs_program.html. Accessed August 20, 2019.

9. Patrick E. Cole, "Lost a computer file? Call on Dr. Norton," *Bloomberg Businessweek*, May 23, 1988, 116.

with either one or two floppy disk drives, depending on the configuration that the customer purchased. The operating system was loaded via a floppy disk—either MS-DOS or CP/M—and then the user could switch disks to run an application program or access the data that they had stored during a previous computing session.

To supplement his explorations of this world, Peter Norton used IBM's *Technical Reference Manual* (1981) to learn what he could about the new IBM system, and he made notes when the manual was unclear or incorrect. His earlier work with computer architecture, hexadecimal notation, and assembly language gave him important insights into the design of the system, which was delivered to customers with little explanation other than a short reference guide.

As Norton later described it, one day in early 1982 he was working with his IBM PC and he accidentally deleted a file from a floppy disk that he was using.[10] Accidentally deleting a file was not that unusual. Like most early PC users, Norton was simply faced with the mundane task of entering the file again, a chore that most DOS users knew well. However, during his exploration of system internals he had learned about how data was stored on the sectors of a floppy disk, and Norton decided to retrieve the lost information by attempting to "undelete" the file. In other words, he tried to restore the original document by locating the information before MS-DOS had completed the task of erasing it. This was possible because the first versions of MS-DOS did not actually erase (or destroy) information on disk when the user issued a Delete (DEL) command. Instead, the internal process for deleting a file was gradual; it involved the marking the file's space on disk as available for reallocation, removing the file's location from the file allocation table, and marking the directory entry for the deleted file so that it no longer appeared when the Directory (DIR) commands was issued. The actual *contents* of the deleted file remained unchanged on disk until they were overwritten, a process that could take hours, days, or weeks, depending on how often the disk was used.

Why did the creators of early versions of MS-DOS choose to leave the deleted information on disk for such a long time? Because early PCs were relatively simple devices, in the tradition of hobbyist machines and mail order kits, and the makers of the operating system were relatively unconcerned with data security. They assumed that if computers were locked and in a safe *location*, they would be safe from data tampering and unauthorized use. (See Figure 7.2 for an early IBM Personal Computer and a locking security device.) By design, the users of a PC should have access to all the features of the computer, including the data on disks that were inserted. There was no concern about the existence of leftover file fragments, as

10. Peter Norton, "Introduction," in Rob Krumm, *Inside the Norton Utilities*, Revised and Expanded (New York: Brady Books, 1990).

long as the operating system had the ability to reuse that space when it was needed for later files. In the era before local area networks and the Internet, few PCs had security measures, passwords, or account log-in requirements of any kind. The first computer viruses had not even arrived on PCs.

Using his knowledge about disk sectors and MS-DOS internals, Peter Norton created a new application program called Unerase that could be used to restore deleted files on a disk that the user had accidentally deleted. The only requirements were that the user had to supply the missing first letter of the file name, and that the user would need to issue the UNERASE command soon after they had deleted the original file. (The longer that users waited, the more likely it would be that MS-DOS would use the unallocated storage area for new information.) Norton recognized how useful the program might be, because people were accident prone when it came to computers, and a tool that could restore files might save many hours of work. Moreover, such a program might have commercial value. Norton later commented on the customer need that the Unerase program was designed to address. "Why did The Norton Utilities become such popular software? Well, industry wisdom has it that software becomes standard either by providing superior capabilities or by solving problems that were previously unsolvable. In 1982, when I sat down at my PC to write Unerase, I was solving a common problem to which there was no readily available solution."[11]

Norton wasn't finished. He continued to develop compact system applications or *utility programs*, which were designed to assist the user with a range of common tasks on IBM PCs. At first, he distributed them among friends and user groups in Southern California, finding other PC users who were intrigued with the new machine but aware of its limitations. In the beginning, he hand-delivered pamphlets about his programs to computer stores and user groups, relying on word-of-mouth advertising. In 1982, he formally established Peter Norton Computing, a small software company dedicated to selling his DOS utility programs. The company began with an investment of $30,000 and was essentially run out of an apartment in Venice, California.[12] His first bundle of the programs was called *The Norton Utilities* version 1.0, and it was designed to support the original MS-DOS 1.x operating system. *The Norton Utilities* sold for $80 in its first iteration—not a small sum. The suite of utilities included the Unerase program, Filefix, Filehide, a printing utility, and 10 other programs. In a review of computer publications, the first advertisement that I could find for *The Norton Utilities* was in *InfoWorld* magazine (October 4, 1982). In the ad, the company's postal address was identified as Wilshire

11. Norton, "Introduction," *Inside the Norton Utilities*, xiv.

12. Cole, "Lost a computer file? Call on Dr. Norton," 116.

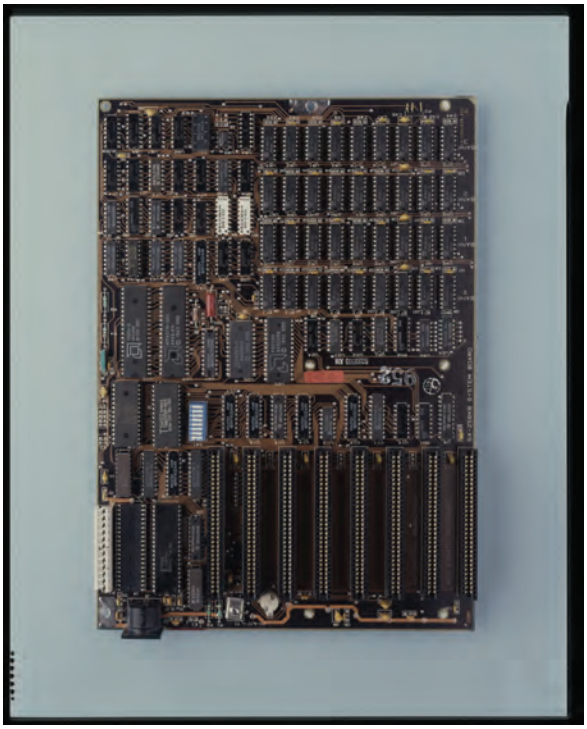


Figure 9.2 **The IBM PC XT motherboard (1983).** (Photo: Herb Bethoney; courtesy of the Computer History Museum)

Boulevard in Santa Monica. As with many early PC software companies, the main sales channel for the product was mail order, a process that typically took a few weeks.

During the next year (1983), a new model of the IBM PC was released. This product, the IBM PC XT, Model 5160, offered users a 10-megabyte (MB) hard drive to store their data, as well as a 5.25" disk drive for installing software and backing up information. (See Figure 9.2.) Microsoft released MS-DOS 2.0 to support the new system and its fixed disk, and they introduced a Unix-like hierarchical file system with subdirectories to help users organize their data. Recognizing the capabilities of the new system, Norton modified his utilities and released *The Norton Utilities* version 2.0, which included a Findfile utility that could help users locate missing files on what seemed like an expansive storage system.

Peter Norton had learned how to create commercial applications for the new IBM PC. But how would other would-be developers learn to create applications for the new system? Could enthusiasm for the IBM PC be turned into a new chapter in the learn-to-program movement?

A little before the IBM PC XT was released, Peter Norton was contacted by Dr. Larry Joel Goldstein (1944–) of Brady Books to write a computer book that would instruct software developers how to programmatically access the advanced features of the new IBM Personal Computer. Goldstein hoped that the primer would be both a "how-to" programming book about writing software and also a comprehensive guidebook to the IBM PC hardware architecture and the internals of MS-DOS. Goldstein had impressive credentials in both technology and publishing. He was a long-time member of the Association for Computing Machinery (ACM), and he had served as a consulting editor for the Robert J. Brady Company since 1980, acquiring programming books about the new microcomputers as they arrived on the scene. Goldstein earned a Ph.D. in Mathematics from Princeton (1967), and he later wrote books about mathematics, programming languages, and data science. Dr. Goldstein also had experience with entrepreneurship, founding two software companies in the 1980s. In other words, connecting with Norton was serendipitous for both men. The resulting computer book, *Inside the IBM PC* (1983), became an early classic in the PC programming field, and it was lauded by programmers and reviewers alike as "an indispensable manual."[13] In fact, the book was much better than a manual, offering personal insights and detailed technical information gained through trial and error. Norton taught new IBM PC programmers the ins and outs of the Intel 8080 processor, how ports and registers worked, how to call the ROM and BIOS services, and how to use functions related to early graphics monitors. The book ran to some 300-pages, and it included valuable behind-the-scenes information—exactly the type that Norton had used to create his utilities. Fortunately for IBM PC developers, Norton was now willing to share many of these techniques with new-to-topic programmers, many of whom were new to computer architecture and software engineering. Norton included sample code in BASIC, Pascal, and assembly language—the three most popular languages on the IBM PC at that time. His demonstration programs were clearly printed in the book, and he also provided a separate 5.25" disk so that users could load and experiment with the programs.

In his acknowledgments for the guide, Norton thanked several people for their technical assistance with the project. He recognized Joseph Capps, Jr., of IBM, as well as Mike Todd, the President of the Capital Personal Computer User Group (CPCUG) in Washington, DC.[14] The user group was one of the first in the PC community, founded in 1982.[15] Norton also thanked two influential employees at

13. For praise in the popular press for the book, see Dan Robinson, "Peter Norton tells all!" *PC Magazine*, September 1983, 557–558.

14. Norton, *Inside the IBM PC*, x.

15. For a useful finding list of contemporary IBM PC user groups, see Susan Hurley, ed. "Club news," *PC Magazine*, February 7, 1984, 351–363.

Microsoft, Alan Boyd and Chris Larson. Alan Boyd started at Microsoft in 1980 and he worked as a product development manager. Boyd had a gift for making industry contacts and deals for the company, and he was involved with several new ventures and acquisitions, both inside and outside of Microsoft.

Chris Larson was the MS-DOS 2.0 product manager, who started working at Microsoft in 1975 at the age of 16, in part because of his connections to Bill Gates from Lakeside School in Seattle. Larson attended college at Princeton University from 1977 to 1981, and then he returned to Microsoft, receiving a small equity share in the company that grew to be extremely valuable over time. Larson was a highly influential contact for Norton, providing assistance at just the time that Norton was searching for inside information about MS-DOS. (The MS-DOS 2.0 product was finalized on August 3, 1983.) Peter Norton's work with Microsoft was also mutually beneficial: Microsoft received an in-depth technical treatment suitable for its new product, which would encourage application development for the new operating system. Norton learned valuable information that would allow him to update his utilities and write a series of authoritative books that would supplement the official documentation. Martin Campbell-Kelley has framed this synergy as "complementary" product development, which benefited both Microsoft and the firms that added value to Microsoft's products.[16]

Peter Norton gradually achieved notoriety in an industry that was beginning to capture the American public's imagination. He began a regular advice column about system utilities for *PC Magazine* in September 1983. Norton Computing also continued to grow, recording $1 million in revenue in 1984, $5 million in 1986, and $25 million in 1989.[17] Although Norton did most of the technical work himself in the early years, he gradually hired business employees, software developers, and writers to assist with product development, including Brad Kingsbury, John Socha, and Stanley Reifel.

John Socha is most remembered for designing and building *Norton Commander* (1986), one of the first text, menu-based file managers that ran on the MS-DOS operating system. The program was especially helpful to new IBM PC users who could use the tool to move, rename, and delete their files using visual cues and procedures, rather than the cryptic DOS commands that users would type at the system prompt. Socha was also a committed academic with fascinating research interests. While working on *Norton Commander*, Socha was also finishing a Ph.D. in Applied Physics at Cornell University. These interests led him to regular work with the ACM and IEEE societies, where he held long-term memberships. Socha's

16. Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog*, 260.

17. William Aspray and James W. Cortada, "Before it was a giant: The early history of Symantec, 1982–1999," *IEEE Annals of the History of Computing* 38, no. 4 (2016): 26–41, here at 34.

interests in communicating with developers also led him to write a regular column for *Softalk* magazine in the early 1980s, as well as writing projects with Microsoft Press, McGraw-Hill, and other publishers.

Assisted by Socha and others, Peter Norton continued to write books and articles, carefully building his brand identity as "the man with the crossed arms and the pink shirt."[18] In 1985, Norton authored *The Peter Norton Programmer's Guide to the IBM PC*, published by Microsoft Press (Figure 9.3), which emphasized this iconic image on the front cover. (The pose was eventually trademarked by Symantec Corporation.) By this time, Peter Norton was actively involved in numerous book projects, each designed to teach aspiring programmers how to create applications that functioned well on IBM PCs and compatibles. The *Programmer's Guide to the IBM PC* was especially authoritative. It contained 20 chapters of advanced information about the IBM PC's system architecture, ROM-BIOS services, MS-DOS internals, and much more. The book achieved a higher profile than Norton's early programming books because it was published by Microsoft Press and it rode the rising fortunes of the new IBM PC platform after the release of enhanced hardware configurations, including the IBM PC XT, IBM PCjr, IBM Portable PC, and IBM PC AT. The second edition of the *Programmer's Guide* was published in 1988, followed by a third edition in 1993. Each pictured the author's iconic "crossed-arms pose" on the front cover. *Popular Computing* magazine wrote about the series, "Whether for advanced programmers involved in developing professional software or for those simply curious about how the PC really works, [this book] is mandatory reading."[19]

The first two editions of the Microsoft Press book sold 500,000 copies, and the third edition also achieved best-seller status, helping to establish the category of "advanced" programming books written by third-party (independent) authors. These substantial sales figures indicate that more than just a few assembly language programmers were peering into the internals of MS-DOS and IBM PCs. Thousands of aspiring developers who had learned the basics of programming using other tools and texts were now turning to professional grade topics. Programming advocates like Peter Norton had "scaffolded" the learn-to-program movement so that it moved from beginning to intermediate to advanced topics, gradually introducing readers

18. Norton cultivated this image as early as 1982, apparently in association with the first edition of The Norton Utilities. His 1983 author biography from *Inside the IBM PC* concludes with the sentence, "Mr. Norton lives on the beach in Venice, California, and always wears his shirt sleeves rolled-up." See Norton, *Inside the IBM PC*, xi.

19. Peter Norton, Peter Aitken, and Richard Wilton, *The Peter Norton PC Programmer's Bible: The Ultimate Reference to the IBM PC and Compatible Hardware and Systems Software* (Redmond, WA: Microsoft Press, 1993). Quote listed is on the inside front cover. The book was retitled to emphasize its status as the *definitive* Peter Norton programming guide (of which there were many).

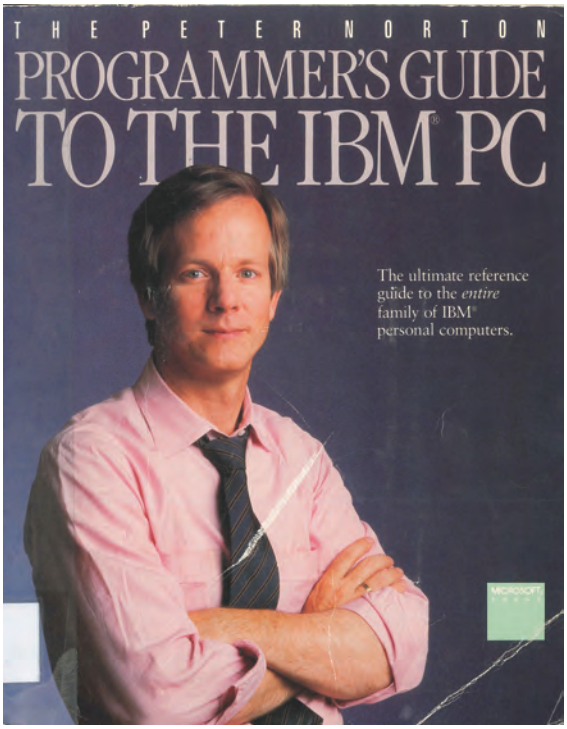


Figure 9.3 **A well-used library copy of *The Peter Norton Programmer's Guide to the IBM PC*, published by Microsoft Press (1985). Norton's pose and pink shirt became an important part of his brand identity in the 1980s.** (Used with permission from Microsoft)

to the commercial opportunities available to them via the expanding family of IBM PCs and compatibles. As anecdotal evidence of this surge in interest, I located the worn copy of *The Peter Norton Programmer's Guide* shown in Figure 9.3 at a public library in Seattle. Its circulation information indicated that it had been checked out hundreds of times over the past three decades.

Peter Norton used assembly language for several of his earliest utilities, but by late 1983 he wrote that he had come to prefer the Pascal programming language for its "lean, crisp nature" as well as its support for structured programming.[20] Gradually, the teams at Norton Computing came to utilize the C language for their work, admiring its speed and close connection to the underlying hardware of PCs.[21] Earlier stalwarts such as FORTRAN, COBOL, and BASIC had

20. Peter Norton, "PC languages: The living and the dead," *PC Magazine*, September 1983, 99–101, here at 99.

21. Norton, "PC languages," 101.

now given way to more efficient high-level languages, at least for professional programmers writing commercial code.

Over the next several years, Norton Computing added several applications to their product line, including Norton Integrator, Norton Disk Doctor, Speed Disk, and Norton Backup. To extend their reach, the company also developed Apple Macintosh, Microsoft Windows, and Unix versions of their software suites, expanding on the original MS-DOS products. Collectively, these companion products should be seen as enhancing the experience of PC users, who bought base systems that were still relatively modest in terms of functionality. Many users realized that they needed to purchase additional software to protect their work and investments in the new platform.

Reflecting the overall success of these products, Peter Norton sold his company to Symantec Corporation in 1990 for $70 million, just 8 years after it was founded in Santa Monica. The acquired company became a division of Symantec, renamed the Peter Norton Computing Group. The merger allowed the employees of Norton Computing to receive equity in a public company without going through an initial public offering (IPO) on their own, and Symantec was able to approximately triple the revenues of Norton Computing's products within a year of the merger.[22] In this way, the Norton brand lived on, offering an expanding suite of software tools to analyze, configure, and maintain PCs. Throughout the 1990s, most Norton products continued to feature an image of their famous founder, his sleeves rolled up and ready to help with whatever problems new users confronted. What is less well-known is the vital role that Peter Norton played in the education of new IBM PC programmers, who needed reliable, well-written resources to help them build commercial software and extend the MS-DOS platform. This support was especially significant between 1982 and 1986, the first years of commercial MS-DOS programming.

9.3 Borland's Turbo Pascal

Another early entrepreneur associated with PC programming tools and utility programs was Philippe Kahn (1952–), a French mathematician who started the software company Borland International with a group of Danish investors in 1983. The new firm considered various products to create and market, but they settled on an intriguing new Pascal compiler that had recently been created by the Danish software developer Anders Hejlsberg (1960–). Hejlsberg was a student at the

22. For this claim, see "Oral history of Gary Hendrix," interviewed by Dag Spicer, November 19, 2014, Computer History Museum, 37.

Technical University of Denmark in Copenhagen, where he studied Electrical Engineering and gained experience with programming, kit-based computers, and early microprocessors. As a first-year student, he met a group of students who had formed a company named PolyData to sell computer products in Copenhagen.[23] As part of his work with them, Hejlsberg created a "Tiny-Pascal" compiler that could fit into 12KB of memory, and PolyData sold it locally under the name Blue Label Software Pascal (later Compas Pascal). When the Danish investment group behind Borland heard about the new product, they licensed the technology from PolyData, thinking that it might be a good fit for the emerging PC marketplace in the U.S.

After the licensing deal, one of the important enhancements that the Borland team made was to create an integrated development environment (IDE) for the Pascal compiler, which simplified the editing and debugging process for programmers and made the overall development process faster. Borland named the new product Turbo Pascal, and the first version of the Pascal compiler was released in 1983 for the CP/M and MS-DOS platforms. Philippe Kahn insisted on selling the product for $49.95 through mail order channels, even though the price was hundreds of dollars less than the market leaders, including IBM Pascal, Microsoft Pascal, and UCSD Pascal. However, the low-cost integrated product was attractive to both novice and experienced programmers alike, and it arrived at exactly the time when the market for IBM PCs and compatibles was expanding. By the end of 1985, a reported 400,000 copies of Turbo Pascal had been sold by Borland.[24] Most of them were purchased by hobbyist and self-taught programmers who were writing simple programs, games, and utilities for the MS-DOS platform. Anders Hejlsberg continued to work on the compiler remotely from Denmark while Philippe Kahn developed a marketing and support footprint for the company in Scotts Valley, California.[25]

The Turbo Pascal language can be considered an early "structured" language, based on a dialect of Pascal popularized by Niklaus Wirth in the book *Algorithms + Data Structures = Programs* (1976).[26] As I noted in Chapter 5, the mid-1980s was an era in PC programming history in which software publishers were under intense pressure to release better, more "structured" languages and tools

23. Details about the creation of Turbo Pascal can be found in "Life and times of Anders Hejlsberg," interview by Barbara Fox, *Behind the Code*, February 1, 2006. https://channel9.msdn.com/Shows/Behind+The+Code/Life-and-Times-of-Anders-Hejlsberg. Accessed July 19, 2019.

24. Rodnay Zaks, *Introduction to Pascal, Including Turbo Pascal*, Second Edition (Alameda, CA: Sybex, 1988), 5.

25. For a summary of Kahn's activities and business strategy, see Jonathan Weber, "Kahn the Barbarian," *Los Angeles Times*, February 23, 1992.

26. Niklaus Wirth, *Algorithms + Data Structures = Programs* (Englewood Cliffs, NJ: Prentice Hall, 1976).

for software development. Turbo Pascal addressed these needs and surpassed the contemporary versions of BASIC and Pascal published by other companies. In addition, Pascal had become one of the leading "learning languages" for Computer Science programs in colleges and universities. Turbo Pascal seemed made-to-order for this market, especially when institutions of higher education began purchasing IBM PCs and compatibles for their computer labs. By 1985, it is estimated that over 400 colleges and universities were using Turbo Pascal for their introductory programming courses.[27] Moreover, the Turbo Pascal IDE was widely appreciated for increasing programmer productivity.[28] Programmers were able to open, edit, run, and debug their programs with intuitive, text-based menus and commands. The text editor also supported innovative WordStar-type keyboard shortcuts that could be used for program editing tasks. Later editions of Turbo Pascal were further enhanced with contextual help features, sample code libraries, and other forms of digital documentation.

Sensing a business opportunity in Turbo Pascal, numerous trade and textbook publishers responded to the interest in Borland's compiler by issuing primers for students and self-taught programmers. These books began arriving in 1985 and continued to be popular through the early 1990s. Strong sellers included *Turbo Pascal: A Problem Solving Approach*, by Elliot B. Koffman (Addison-Wesley); *Introduction to Pascal*, by Rodnay Zaks (Sybex); *Turbo Pascal*, by Walter J. Savitch (Benjamin Cummings); *Getting the Most from Turbo Pascal*, by James T. Smith (Blaise Computing); *Complete Macintosh Turbo Pascal*, by Joseph Kelly (Scott, Foresman); *PC Magazine Turbo Pascal 6.0 Techniques and Utilities*, by Neil J. Rubenking (Ziff Davis Press); and *A Second Course in Computer Science with Pascal*, by Daniel D. McCracken (Wiley). The best of the primers focused on the unique features of the Turbo Pascal IDE as well as the programming language. However, some publishers simply updated earlier textbooks covering Standard Pascal or UCSD Pascal, presenting little that was new about the Turbo Pascal platform. They did this because they had not yet adapted to the rapid revision schedule of PC software programs. Borland and Microsoft routinely upgraded their software every year to 18 months, but academic publishing schedules moved slower, until they realized the importance of regular updates.

Daniel McCracken's textbook book is an excellent example of a Turbo Pascal primer that was designed specifically for the academic marketplace. *A Second Course in Computer Science with Pascal* focused on using Turbo Pascal to gain

27. Zaks, *Introduction to Pascal*, 5.

28. For a review of Turbo Pascal 3.0 making this claim, along with editorial commentary, see Mark Bridger, "Software review: Turbo Pascal 3.0: An update on Borland's compiler," *Byte* 11, no. 12 (February 1986): 281–286.

experience with fundamental computing concepts including arrays, sets, stacks, recursion, linked lists, sorts, searches, and graphs. These were topics that an undergraduate Computer Science major would typically encounter in a "data structures and algorithms" course, and the book provides some evidence that new PC programming tools were making an inroad into the academic curriculum. McCracken was President of the ACM from 1978 to 1980. (See Figure 3.9.) He was first introduced in *Code Nation* as the best-selling author of FORTRAN tutorials, which popularized the learn-to-program movement in mainframe and minicomputer contexts (see Chapter 3). In his later years, McCracken taught Computer Science at The City College New York and published programming primers and computer science textbooks. In a fascinating open letter to the ACM membership in January 1987 (a month before his Turbo Pascal book was published), McCracken described how fundamental topics in computer science needed constant experimentation and revision in the classroom. He also emphasized the value of introducing "advanced" topics in programming courses such as advanced data types, recursion, and software engineering principles.[29] Fundamentally, McCracken wanted to encourage "hands on" computing courses. "Programming has to be learned through the fingertips... without it, we are teaching piano in a lecture course."[30] From the perspective of the learn-to-program movement, Turbo Pascal books like McCracken's provided an important extension to the movement's goals, and they helped industry experts teach a wide range of skills to new audiences.

Although Turbo Pascal was a low-priced, "entry level" product in terms of market segmentation (it was sold primarily to hobbyists and students, not commercial software developers), the compiler was admired throughout the industry for its speed and innovative design. Much of this was due to the continuing influence of Anders Hejlsberg, who became principal engineer at Borland, moved to California, and contributed many new features to the product and its successors. These included the introduction of object-oriented programming functionality in Turbo Pascal 5.5 (1989) and the release of Delphi (1995), a powerful rapid application development tool for Windows. In 1996, Hejlsberg left Borland for Microsoft, where he became architect for the Visual J++ development system, a contributor to the .NET Framework, and chief designer of the C# programming language.[31]

29. Daniel D. McCracken, "Viewpoint: Ruminations on Computer Science curricula," *Communications of the ACM* 30, no. 1 (January 1987), 3-5.

30. McCracken, "Viewpoint," 3.

31. For more about these systems, see the Microsoft Developer Network website https://channel9.msdn.com/Shows/Behind+The+Code/Life-and-Times-of-Anders-Hejlsberg. For the C# language specification and commentary, see Anders Hejlsberg, Scott Wilamuth, and Peter Golde, *The C# Programming Language* (Upper Saddle River, NJ: Addison-Wesley, 2003).

Hejlsberg is considered to be one of the most influential language designers in the computer industry.

9.4 Ray Duncan's *Advanced MS-DOS*

If Turbo Pascal encouraged aspiring PC programmers to create some of their first applications and utilities for the MS-DOS platform, Ray Duncan's *Advanced MS-DOS* (1986) gave experienced assembly language and C developers the encouragement that they needed to go deeper into the architecture of MS-DOS and build commercial applications that were elegant, efficient, and robust. As the MS-DOS platform reached maturity (MS-DOS versions 3.0 to 6.0), Duncan became the programmers' programmer, a reputation he gained through innovative work as a software publisher, book author, and columnist. Like other well-known programmer-authors, Duncan's compelling prose wove advanced technical content with clear and practical instruction, encouraging new-to-topic developers to try their hand at building real-world PC applications. Through his influence, the learn-to-program movement expanded into the realm of operating systems and system services, at a time when quality reference materials and tutorials were in relatively short supply.

Raymond G. Duncan was born in 1952 and was raised in Southern California. He received a B.A. in Chemistry in 1973 from the University of California, Riverside, and an M.D. from UCLA in 1977.[32] After graduating from medical school, Duncan began a career in health care, specializing in pediatrics and neonatology (the medical care of newborn infants). (See Figure 9.4.) He developed a long-standing relationship with Cedars-Sinai Medical Center in Los Angeles, where he served as a fellow in neonatology and a member of the teaching faculty. Later in his career, he took on additional administrative duties, serving as the Chief Technology Officer of the Cedars-Sinai Health System.

Duncan had been fascinated with electronics and computers from an early age, receiving hands-on experience from an uncle who was an electronics technician and radio operator for the U.S. Forest Service. To earn extra money in college, Duncan worked as a part-time software developer in medical labs in the LA area, learning about systems programming from experiments with a 16-bit Raytheon 703 computer system that had been customized for pathology work in local medical labs. Duncan's experiments with instruction sets, assemblers, disk storage, and teletype terminals were almost entirely self-directed, with occasional support from a

32. A comprehensive biography of Ray Duncan has not been written. The sketch that follows comes from published information in Duncan's books, as well as email correspondence between Duncan and the author in June and July, 2019.

Figure 9.4 **Dr. Ray Duncan worked as a physician and specialist in neonatology, while also developing advanced expertise in microcomputer operating systems, programming languages, and compilers.** (Photo courtesy of Ray Duncan, *c.* 1990)

Raytheon instruction manual and a few early computer books, including Donald Knuth's *Art of Computer Programming* series.[33]

When early microcomputers hit the scene, Duncan experimented with an MITS Altair 8800 powered by an Intel 8080 CPU.[34] He later purchased an IMSAI 8080 microcomputer and began writing low-level programs for it, gaining experience with the new CP/M operating system. CP/M, a product of Digital Research, Inc., had been initiated by Gary Kildall in 1974 as a compact control system for micros based on the Intel 8080 and 8085 processors. Although CP/M was modified and improved over the years by Digital Research, the product did not maintain its early lead in the microcomputer market and essentially disappeared from view by the mid-1980s. However, Duncan understood the bourgeoning system well, and the

33. Unpublished email correspondence between Ray Duncan and the author, June 2019.

34. Ray Duncan, *Advanced MS-DOS*, 469.

experience was helpful to him as he worked with later systems' software, including Unix, MS-DOS, and OS/2.

Duncan preferred to stay "close to the metal" in his coding work, and he quickly adapted the systems he experimented with for use in healthcare. For example, he wrote programs for the neonatal unit at Cedars-Sinai that computed growth charts, calculated intravenous (IV) drips, and allowed the medical staff to view and print reports on time-sharing terminals. Duncan also became interested in the Forth programming language, because it seemed well-suited for device control and data acquisition tasks in low-memory environments. Forth was designed by Charles "Chuck" H. Moore in 1970 and popularized by Moore, Elizabeth Rather, and Donald Colburn in the following years.[35] (Leo Brodie also wrote a popular book about Forth in 1981 that piqued the interest of many programmers.[36]) Duncan experimented with a public domain version of Forth on CP/M, and he enhanced it for several microcomputer platforms. In 1977, Duncan formed Laboratory Microsystems, Inc. to design and sell Forth-related products, advertising his software in magazines such as *Computer Shopper*. When the original IBM Personal Computer came out, Duncan was the first Forth compiler publisher to release a product for the new system (1982).[37] By 1983, he had designed Forth products and cross-compilers for the Zilog Z-80, Intel 8088, Intel 8086, and Motorola 68000 microprocessors.[38] As part of this work, Duncan studied computer architecture closely, learning about how to integrate his compilers with CP/M, MS-DOS, Apple DOS, and other operating systems.

While continuing his duties as a pediatric resident at Cedars-Sinai Medical Center, Duncan experimented with writing technical articles about PCs, compilers, and assemblers. Between 1982 and 1983, he submitted essays to *Dr. Dobb's Journal*, *Softalk/PC*, and *PC Tech Journal*—three respected trade magazines with interesting content for the professional programming community. He also published a book through Laboratory Microsystems entitled "MS-DOS Internals," which explained how to use the ROM BIOS and operating system functions of MS-DOS. Duncan developed a reputation for exploring the inner worlds of storage devices, system services, memory management, and device drivers—topics that went well beyond

35. For a history, see Elizabeth D. Rather, Donald R. Colburn, and Charles H. Moore, "The evolution of Forth," in *History of Programming Languages-II*, ed. Thomas J. Bergin and Richard G. Gibson (New York: ACM Press, 1996), 625–670, here at 645.

36. Leo Brodie, *Starting FORTH: Introduction to the FORTH Language and Operating System for Beginners and Professionals* (Englewood Cliffs, NJ: Prentice Hall, 1981). By 1995, the primer had sold a reported 110,000 copies, suggesting the language enjoyed a popular following.

37. Rather, Colburn, and Moore, "The evolution of Forth," 642.

38. Advertisement, Laboratory Microsystems, Inc., in *FORTH Dimensions* (published by FORTH User's Group), vol. 5, no. 2 (July–August, 1983), 34.

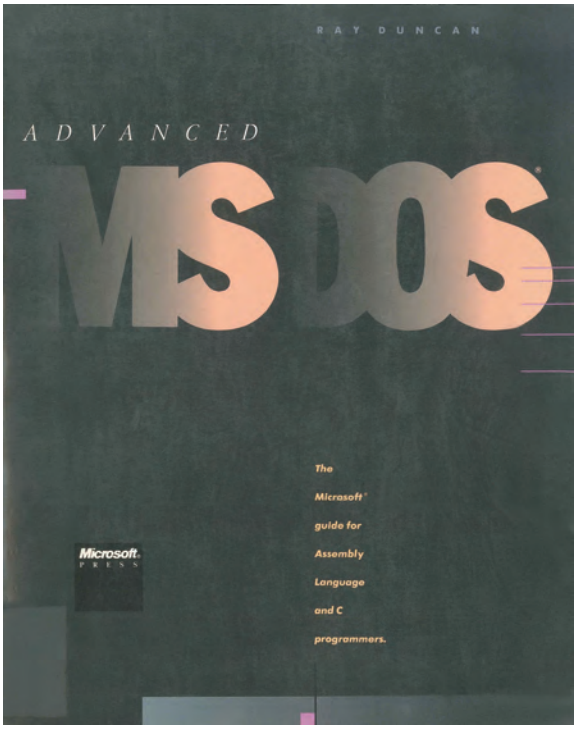


Figure 9.5 **Ray Duncan's *Advanced MS-DOS* (1986).** (Used with permission from Microsoft)

language fundamentals or the application of simple algorithms. His writing had the gravitas of peer-reviewed journal articles from the Computer Science or Software Engineering fields, but it also welcomed self-taught programmers who were just getting their feet wet with systems programming. In the coming years, Duncan became a regular columnist for *PC Magazine* ("Power Programming"), *Embedded Systems Programming* ("When in ROM"), and *Microsoft Systems Journal*.

In late 1985, Duncan received a call from Claudette Moore, an acquisitions editor at Microsoft Press who had seen and appreciated Duncan's self-published book, "MS-DOS Internals."[39] The conversation resulted in the publication of a new book for Microsoft Press, *Advanced MS-DOS*, a 468-page introduction to DOS systems programming. (See Figure 9.5.) Duncan's reference began with a genealogy describing the history of MS-DOS and its relationships to other operating systems. The progression culminated with MS-DOS version 3.2, released in mid-1986. Duncan's approach to writing revealed interests in business, history, and engineering. He

39. Unpublished email correspondence between Ray Duncan and the author, June 2019.

avoided evangelizing for Microsoft or taking sides in the era's vigorous "platform wars." Instead, he emphasized the rich connections among the era's microcomputer and minicomputer operating systems:

> From the programmer's point of view, the current versions of MS-DOS (versions 2 and 3) are robust, rich, and powerful development environments. A broad selection of high-quality programming tools is available from both Microsoft and other software houses. Porting existing applications into the MS-DOS environment is relatively simple, since the programmer can choose to view MS-DOS as either a superset of CP/M or a subset of UNIX.[40]

The book encouraged assembly language and C programmers to use the internal features of MS-DOS to create applications, filters, and installable device drivers. There were sections on the internal structure of DOS; how the programming environment functioned from the point of view of a developer; how to call MS-DOS interrupt handlers; and reference materials documenting the MS-DOS and IBM PC BIOS functions. The book emphasized assembly language, and the reference sections allowed readers to see at a glance what the contents of the microprocessor's registers looked like before and after major function calls. Most of the programs and code fragments were written using the Microsoft Macro Assembler (MASM) version 4.00 compiler. This product was initially released with the first version of MS-DOS in 1981, and it was updated for each new version of MS-DOS and OS/2 to integrate new operating system features. (IBM also distributed MASM under the name IBM Macro Assembler for the early versions of PC-DOS.) The C language source listings were developed using the Microsoft C Compiler version 3.00.

Advanced MS-DOS did not teach assembly language or C programming concepts, but it relied on an earlier exposure to these languages. This represented a potential problem for some readers, who could quickly get in over their heads with the complexity of the source listings and the steps required to build commercial applications. Indeed, there was more space devoted to MASM than to C in the book, because assembly language made the process of calling MS-DOS functions easier and more efficient. Duncan did recommend some resources for learning to program, however, such as Robert Lafore's *Assembly Language Primer for the IBM PC & XT* (1984).[41] In 1991, Duncan also published his own guide to assembly

40. Duncan, *Advanced MS-DOS*, 4.

41. Robert Lafore, *Assembly Language Primer for the IBM PC & XT* (New York: New American Library, 1984).

language for intermediate to advanced programmers, with advice about writing modular, maintainable, and well-documented programs.[42]

In MS-DOS programming, most of the system services were invoked by an enigmatic microprocessor software interrupt known as "21H." This family of function calls allowed a programmer to inspect disk directories, create or delete files, read or write records within files, set the system date and time, allocate memory, and perform other tasks.[43] Learning how to use these interrupts was a major challenge of operating systems programming, and Duncan devoted several chapters and reference sections to them. Another demanding task was learning how to create device drivers, which are the modules of an operating system that control the computer's hardware. Microsoft introduced installable device drivers in MS-DOS version 2.0, and they allowed the user to customize and configure an IBM PC for a wide range of peripheral devices, such as new hard disks or modems.[44] MS-DOS device drivers are interfaced to the hardware-independent DOS kernel through a clearly defined scheme of function codes and data structures. Like the other components of MS-DOS, Duncan described the required elements using conceptual drawings, concise technical prose, and working assembly language routines that programmers could adapt to their own needs. If you worked at a company making peripherals or device drivers in the early years of the PC industry, it is likely that your team had a copy of Duncan's *Advanced MS-DOS* on their bookshelves. This primer and reference provided essential information that was hard to come by via other routes.

Advanced MS-DOS was well reviewed and the book seemed to edify new-to-topic programmers and experienced developers alike. *Byte* magazine emphasized the literary qualities of the book. "*Advanced MS-DOS Programming* exemplifies how a highly technical book can be both informative and readable."[45] In 1988, the book was updated with a second edition, covering newer compilers and the Intel 80386 microprocessor.[46] The sample programs were written for MS-DOS through version 4 using Microsoft Macro Assembler version 5.1 and the Microsoft C Compiler version 5.1. (Figure 9.6 shows the MASM Programmer's Guide that a typical user would

42. See Ray Duncan, *Power Programming with Microsoft Macro Assembler* (Redmond, WA: Microsoft Press, 1991).

43. Duncan, *Advanced MS-DOS*, 274.

44. Duncan, *Advanced MS-DOS*, 222.

45. John D. Unger, "Advanced MS-DOS review," *Byte Extra Edition*, vol. 11, no. 11 (1986), 23.

46. Ray Duncan, *Advanced MS-DOS: The Microsoft Guide for Assembly Language and C Programmers*, Second Edition (Redmond, WA: Microsoft Press, 1988). At Microsoft Press, I was the technical editor for the second edition of the book and I also prepared the companion disk with source code listings. Mark Zbikowski continued to provide advanced technical support from the MS-DOS group inside Microsoft.

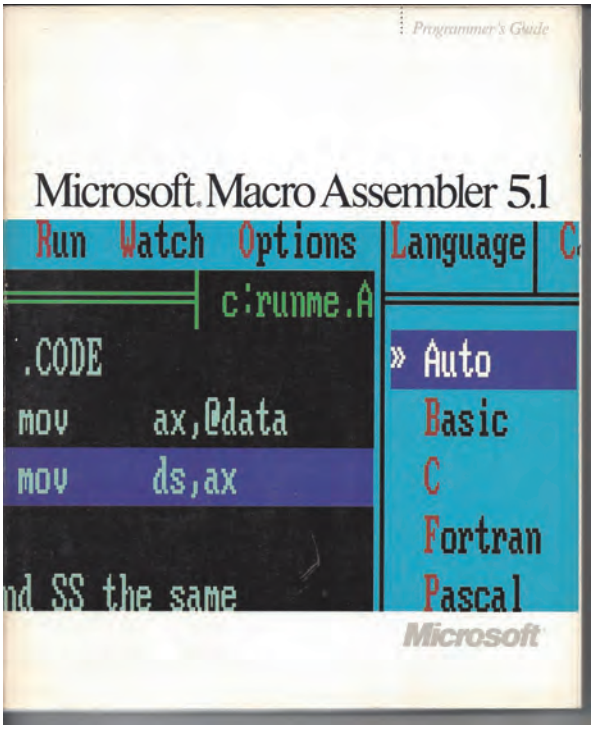


Figure 9.6 **Cover of the Microsoft Macro Assembler 5.1 Programmer's Guide, for IBM PCs and compatibles running the MS-DOS operating system (*c.* 1987).** (Used with permission from Microsoft)

have had access to. The text would supplement the material presented in Duncan's *Advanced MS-DOS*.)

At the time of the second edition, Microsoft was engaged in a power struggle with IBM, as the two companies sought to jointly develop the new multitasking, graphical operating system called OS/2. Ignoring the politics of the issue, Duncan became intrigued with the underlying technologies of the system, and he soon released another full-length title devoted to the OS/2 system kernel. *Advanced OS/2 Programming* (1989) focused on the substructures of OS/2 and the multitasking capabilities of the operating system.[47] Duncan did not cover Presentation Manager in his book, the new graphical windowing environment that displayed and managed information for users. (He left this task to Charles Petzold, the author of

47. Ray Duncan, *Advanced OS/2 Programming: The Microsoft Guide to the OS/2 Kernel for Assembly Language and C Programmers* (Redmond, WA: Microsoft Press, 1989). Eric Stroo was the book's project editor and I continued as the book's technical editor. Claudette Moore, Duncan's first acquisitions editor at Microsoft Press, continued on in a new role as Duncan's literary agent.

Programming the OS/2 Presentation Manager.) However, IBM and Microsoft gradually dissolved their joint partnership in OS/2, and by the early 1990s Microsoft put the majority of its resources behind the Microsoft Windows platform. This included the development of Windows NT, a scalable, multiprocessing operating system that would soon become the basis for the company's enterprise and client/server initiatives. (For more about Windows and Windows NT, see Chapters 10 and 11.)

9.5 *The MS-DOS Encyclopedia*

In computer book publishing, the era of MS-DOS development probably has no more comprehensive account than the massive *MS-DOS Encyclopedia* (1988), published by Microsoft Press and shepherded by general editor Ray Duncan. A monumental project, the final hardback edition of the *Encyclopedia* came in at 1570 pages and took several years to produce. In a 1988 Foreword to the tome, Bill Gates, the Chairman of Microsoft Corporation, wrote that MS-DOS was the most popular piece of software in the world, running on more than 10 million IBM PCs and compatibles, and serving as the platform for at least 20,000 applications.[48] The company's goal with *The MS-DOS Encyclopedia* was, as Gates put it, to "provide the most thorough and accessible resource available anywhere for MS-DOS programmers."[49] The effort to produce the book was indeed substantial. It involved sizeable teams of managers, editors, proofreaders, editorial assistants, production personnel, software experts, and technical writers. The cohort of outside technical experts (called "contributors" in the book's front matter) included Ray Duncan, Steve Bostwick, Keith Burgoyne, Robert A. Byers, Thom Hogan, Jim Kyle, Gordon Letwin, Charles Petzold, Chip Rabinowitz, Jim Tomlin, Richard Wilton, Van Wolverton, William Wong, and JoAnne Woodcock. Sixty-four "technical advisors" were also listed, including the Microsoft luminaries Paul Allen, Steve Ballmer, Bill Gates, Chris Larson, Marc McDonald, Tim Paterson, Bob O'Rear, and Mark Zbikowski.[50]

The only woman in the group of featured "contributors" was JoAnne Woodcock (1944–2014), a senior editor at Microsoft Press who I first introduced in Chapter 6. Woodcock was responsible for the essay about the historical development of MS-DOS, an assignment that grew out of her deep experience editing user's guides for PC operating systems, as well as earlier work at the *Encyclopedia Britannica*. In the late 1980s, Woodcock was also the co-author of two computer books about the Xenix and Unix operating systems, which encouraged new users to become productive

48. Bill Gates, "Foreword," in *The MS-DOS Encyclopedia*, ed. Ray Duncan (Redmond: Microsoft Press, 1988), xiii.

49. Gates, "Foreword," *The MS-DOS Encyclopedia*, xiv.

50. Duncan, *The MS-DOS Encyclopedia*, vii–viii.

with the operating system software in business and corporate contexts.[51] Woodcock also co-authored a pioneering book on word processing (with Peter Rinearson) entitled *Word Processing Power with Microsoft Word*.[52] This guide explained how to use mail merge, style sheets, macros, and other advanced features of Microsoft Word for MS-DOS version 5 (the character-based version of Microsoft's word processor). Woodcock's last job title at Microsoft Press was Master Writer/Editor, a position created to honor her many contributions to technical publishing and editing. In 1994, she published *The Ultimate MS-DOS Book*, a comprehensive user guide covering the MS-DOS system that she had poked, prodded, and explained for so many years.[53] This book distinguished itself from the competition by focusing on the problems and opportunities that upgraders to MS-DOS 6.0 and 6.2 might encounter. Like the other titles in the Microsoft Press *Ultimate* series, this guidebook was thoroughly illustrated. Woodcock used the book to emphasize the points of friction or "sticky wickets" that users might encounter when they upgraded existing systems or explored new features.

JoAnne Woodcock was widely recognized as an authority on operating systems, but was she the only woman credited in *The MS-DOS Encyclopedia* Looked at comprehensively, the front matter for the book lists 79 technical experts that were consulted for the project (both "contributors" and "technical advisors").[54] Examining the list of names for traditional gender attributions, it is evident that approximately 94% of the credited technical experts were male (74 of 79 participants). In terms of employment, most of the consultants were senior engineers, system architects, and software developers at Microsoft in the MS-DOS group, with additional authors and authorities from the software industry. This gender imbalance is typical of the U.S. software industry in the late 1980s, with men outnumbering women on engineering and software development teams at rates of 3-to-1 or higher.[55] The extraordinary imbalance on this reference work may be a result of the involvement of numerous

51. See JoAnne Woodcock and Michael Halvorson, eds., *Xenix at Work* (Bellevue, WA: Microsoft Press, 1986); JoAnne Woodcock, Michael Halvorson, and Robert Ackerman, *Running UNIX: An Introduction to SCO UNIX System V/386 and XENIX Operating Systems* (Redmond, WA: Microsoft Press, 1990).

52. Peter Rinearson and JoAnne Woodcock, *Word Processing Power with Microsoft Word*, Third Edition (Redmond, WA: Microsoft Press, 1989).

53. JoAnne Woodcock, *The Ultimate MS-DOS Book* (Redmond, WA: Microsoft Press, 1994).

54. Duncan, *The MS-DOS Encyclopedia*, vii–viii.

55. In her study of women's participation in computing, Janet Abbate puts the rate of participation of women in programming jobs at 33% in 1988 using data from the U.S. Department of Labor. See Janet Abbate, *Recoding Gender: Women's Changing Participation in Computing* (Cambridge, MA: The MIT Press, 2012), 3.

high-ranking technical managers and executives who had been at the company for some time and hoped to participate in the project. Despite the scarcity of female contributors, however, there were several women listed as technical advisors for the book, including Rachel Duncan, Estelle Mathers, Betty Stillmaker, JoAnne Woodcock, and Natalie Yount.

The masthead of *The MS-DOS Encyclopedia* also lists 54 employees who participated on the project from the Microsoft Press division. (See Figure 9.7 for a group photo including many of them.) Interestingly, approximately half of these employees (48%) were women, including many in senior leadership roles. These managers included Dorothy L. Shattuck (Senior Editor), Patricia Pratt (Editorial Director), Sally Brunsman (Special Projects Editor), Brianna Morgan (Copy Chief), and Susan Lammers (Editor-in-Chief). Rounding out the list of senior managers were David Rygmyr (Senior Technical Editor), James Brown (Marketing and Sales Director), Christopher Banks (Director of Production), and Min S. Yee (Publisher). The relatively equal gender distribution of managers and employees on the publishing side of the business was typical at Microsoft Press and in many American publishing houses in the 1980s. This points out an interesting caveat about the gender imbalance that scholars typically find in high-tech work groups. While men continued to be overrepresented in the engineering and management positions at software companies in the 1980s and 1990s, women were sometimes well represented in affiliated areas, including product support, customer service, publishing operations, corporate communications, and computer training/education. While many of these areas did not pay as well as the engineering groups in terms of base salary, they could offer tangible benefits to employees, including stock options and more flexible schedules than the coding-intensive positions allowed. Future historians of computing should try to make these affiliated areas and their employment dynamics more visible. Software companies could assist in the process by providing limited access to corporate records and archival materials, with sensitive information redacted as appropriate.

9.6 MS-DOS Sample Code

Like *Advanced MS-DOS*, *The MS-DOS Encyclopedia* contained significant amounts of sample code so that curious programmers could learn advanced features and integrate them into projects. Most routines came in the form of *skeleton code*, the fragmentary code blocks that could easily be expanded into substantial routines and modules. For example, the following code block was written in Microsoft Macro Assembler version 4.00 to demonstrate how to create a new file named MEMO.TXT in the \LETTERS directory on a disk labeled "C". The routine was designed to demonstrate how the file and record management function 3CH operated under

Figure 9.7 **Microsoft Press personnel at a company retreat, mid-1987. At the time this photo was taken, many of the organization's staff members were working on *The MS-DOS Encyclopedia*, a task that occupied the entire publishing house. Publisher Min S. Yee is standing on the left of the image. Editor-in-Chief Susan Lammers holds the division's sign near the center.** (Photo courtesy of Michael Halvorson)

MS-DOS version 3.2. If a file with the same name already existed in the directory, it would be "truncated" to zero length and opened for input. Explanatory comments appear after the semicolon (;) character in the column on the right side of the code block.

```
fname    db      'C:\LETTERS\MEMO.TXT',0
fhandle  dw      ?
                 .
                 .
                 .
                 mov     dx,seg fname    ; DS:DX = address of
                 mov     ds,dx           ; pathname for file
                 mov     dx,offset fname
                 xor     cx,cx           ; CX = normal attribute
                 mov     ah,3ch          ; Function 3CH = create
                 int     21h             ; transfer to MS-DOS
                 jc      error           ; jump if create failed
                 mov     fhandle,ax      ; else save file handle[56]
```

56. Duncan, *The MS-DOS Encyclopedia*, 252.

Most of the programming sections ended with two or three examples in this style to demonstrate how the internal features of the operating system worked.

The MS-DOS Encyclopedia was originally planned to be just the first of several comprehensive reference projects that documented how the major software platforms operated in the PC industry. However, the scope of the project proved daunting for Microsoft and it incurred significant costs. In addition, there was one false start in which early volumes needed to be recalled due to concerns about the accuracy of some technical material and worries that portions of the original MS-DOS source code were being released inadvertently to the public. To address the issue, Microsoft freed up several members of the MS-DOS development teams to carefully review flagged sections, removing information that was either inaccurate or proprietary.[57] Ray Duncan's oversight and influence eventually resolved any outstanding issues.

The 1988 edition of *The MS-DOS Encyclopedia* emerged as a definitive and beautifully produced volume, but it would be the last encyclopedia project of this scope in the computer book industry. For one thing, the hardcover's $134.95 price tag ($69.95 softcover) put the volume beyond the reach of most casual users or programmers. Industry experts and journalists certainly found the book to be invaluable, but the project required a huge investment of internal resources that could not be recouped. To this day, however, *The MS-DOS Encyclopedia* remains the definitive resource for programmers and computer scientists who want to understand how the early versions of MS-DOS came to be, functioned internally, and relate to each other in minute detail.[58] It was perhaps Ray Duncan's most significant contribution to the PC programming world.

9.7 Technology Diffusion

Although MS-DOS programming began as an obscure commercial activity in 1981–1982, it gained momentum as more powerful hardware arrived and the IBM PC platform widened its reach to include new customers. Entrepreneurs like Peter Norton developed utilities for MS-DOS that helped users safeguard their data and become more productive in business environments. A brief history of *The Norton*

57. An interesting blog about the review process was written by one of the Microsoft software developers who was involved with the work. See Larry Osterman, "Does anyone remember the original MS-DOS Encyclopedia? *Microsoft Developer Network*, June 14, 2004. https://blogs.msdn.microsoft.com/larryosterman/2004/06/14/does-anyone-remember-the-original-ms-dos-encyclopedia/. Accessed August 26, 2019.

58. Harvey Deitel of Boston College adapted sections of the *Encyclopedia* for his popular textbook about operating systems, a move followed by other academics and journalists. See H. M. Deitel, *Operating Systems*, Second Edition (Reading, MA: Addison Wesley, 1990), 629–668.

Utilities has demonstrated how dramatically a company's fortunes might have risen when they found the right audience for their applications. However, an overlooked part of this story is the role that book programmer-authors played in the popularization of the MS-DOS platform. The fact that Peter Norton's *Programmers Guide to the IBM PC* sold over 500,000 copies in its first two editions (1985 and 1988) is a testament to the impact that skilled authors and entrepreneurs could made on the early system, when users were scrambling to learn techniques related to the new hardware and software. Ray Duncan's *Advanced MS-DOS, Advanced OS/2,* and *MS-DOS Encyclopedia* also provided helpful reference materials for aspiring DOS developers who were trying to master the inner workings of the operating system for hobbyist or commercial use. At first, PC programmers attempted these tasks using assembly language (Microsoft Macro Assembler) but as time passed they experimented with high-level tools, including QuickBASIC, Turbo Pascal, Forth, and C. When the Microsoft Windows platform gained momentum in the early 1990s, developers gradually shifted away from MS-DOS programming to object-oriented tools that worked well in graphical systems.

Using terminology from the history of technology, we might describe this expanding list of products and transitions as the process of "diffusion" that takes place when a new technology is propagated across society. As other scholars have noted, this diffusion process usually takes place in stages, and it sometimes involves an accompanying discourse that is visible in mass media, trade publications, scholarly journals, trade shows, and other public activities.[59] During the learn-to-program movement, the primary media about the MS-DOS platform were computer books, technical magazines, and corporate communications from technology firms. These taught new-to-topic developers how to build commercial applications, an activity that richly rewarded some of the era's entrepreneurs. Many of the authors of computer books and technical articles were men and women who had earlier experiences in mainframe computing, science, medicine, or journalism. Despite the overrepresentation of men in these fields, there were also some women who wrote about programming and microcomputer and minicomputer operating systems, including JoAnne Woodcock. A particularly conducive environment for women seems to have been technical publishing houses like Microsoft Press, where approximately half of the employees were women. These exceptions to the traditional pattern of male dominance in engineering and technology fields deserves more study.

59. See Margaret S. Elliott and Kenneth L. Kraemer, *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing* (Medford, NJ: American Society for Information Science and Technology, 2008), 5.

In the next chapter, I will continue my analysis of PC programming communities in the 1980s and 1990s, focusing on the C and C++ programming languages, and the tools and techniques that enabled PC programmers to master complex graphical operating systems, such as Microsoft Windows and the Apple Macintosh. I'll also analyze the writings of several fascinating programmer-authors who wrote successful tutorials, including Brian W. Kernighan, Dennis Ritchie, Al Kelley, Ira Pohl, Mitchell Waite, Dan Gookin, and Charles Petzold. I'll discuss the beginnings of the C language on the PC platform, and the eventual complexities of C/C++ programming under Windows that grew so severe they evoked comparisons to a much earlier period of uncertainty—the software "crisis" of the 1960s that threatened to stifle the electronic computer revolution.

10 C Programming Nation: From Tiny C to Microsoft Windows

"C is a relatively 'low level' language. This characterization is not pejorative; it simply means that C deals with the same sort of objects that most computer do, namely characters, numbers, and addresses."

Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language* (1978)[1]

If at first you find Windows programming to be difficult, awkward, bizarrely convoluted, and filled with alien concepts, rest assured that this is a normal reaction. You are not alone.

Charles Petzold, *Programming Windows 3.1* (1992)[2]

When *The MS-DOS Encyclopedia* was published by Microsoft Press in 1988, the editors of the reference printed sample code for MS-DOS applications in assembly language, Microsoft C, and Microsoft QuickBASIC.[3] The encyclopedia also included detailed information about batch file programming, which it described as a useful method for performing sequences of frequently used commands without having to retype them. These four "languages" were the most popular tools for creating applications and utilities in the late 1980s, the heyday of DOS programming on IBM personal computers (PCs) and compatibles. Over the coming decade, however, graphical operating systems such as Microsoft Windows and OS/2 would push the C language to center stage as the most important tool for application development on PCs. For programmers with commercial aspirations, learning C became

1. Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language* (Englewood Cliffs, NJ: Prentice-Hall, 1978), 1.

2. Charles Petzold, *Programing Windows 3.1: The Microsoft Guide to Writing Applications for Windows 3.1*, Third Edition (Redmond, WA: Microsoft Press, 1992), 10.

3. Ray Duncan, ed., *The MS-DOS Encyclopedia* (Redmond, WA: Microsoft Press, 1988), xviii.

a priority. But how should this be attempted in the context of new PC platforms? What software and learning resources were available, and how would novice programmers prepare themselves for the rigors of object-based programming in the Windows or Macintosh environments?

This chapter surveys the development of the C programming language on PCs, and the wide range of learning resources that new-to-topic developers had access to as they built their C programming skills. I emphasize the role that primers, reference guides, and technical articles played in the diffusion of C programming competences, because these media followed the same path that BASIC and MS-DOS resources traveled—they exploited a world of print to communicate with students, especially before the commercial Internet changed dissemination patterns in the mid-1990s. In terms of authors and entrepreneurs that taught C programming skills (listed alphabetically), I'll examine the work of Dan Gookin, Augie Hansen, Samuel P. Harbison, Thom Hogan, Allen I. Holub, Al Kelley, Brian W. Kernighan, André LaMothe, Donald Martin, Kurt Matthies, Ira Pohl, Stephen Prata, Jeff Prosise, Jeffrey Richter, Dennis Ritchie, Guy L. Steele, Jr., and Mitchell Waite. These software developers all prepared learning resources for C and C++ developers in the 1980s and 1990s. I'll also discuss the contributions of Charles Petzold, an award-winning programmer and author who was among the most successful at teaching complex Windows and OS/2 programming techniques. Collectively, this group taught millions of programmers the fundamentals of C, Windows, OS/2, and Macintosh programming, contributing to the success of the learn-to-program movement in its commercial and corporate manifestations.

10.1 The C Language

The early history of the C programming language is relatively well known. The language itself came into being during the years 1969–1973, in parallel with the early development of the Unix operating system.[4] Ken Thompson of Bell Laboratories created a language called "B" (derived from Martin Richards's basic combined programming language [BCPL]), which Thompson planned to use to write system utilities for early versions of Unix. (See Figure 10.1.) As a high-level language, B was presumably easier to use than assembly language to write system tools, and it was hoped that the new language would help the Bell Labs teams save time. The next contributor to the language was Dennis Ritchie, also an employee at Bell Labs, who turned B into the C language between 1971 and 1973. Ritchie

4. Dennis M. Ritchie, "The development of the C programming language," in *History of Programming Languages—II*, ed. Thomas J. Bergin and Richard G. Gibson (New York: ACM Press, 1996), 671–698, here at 672.

Figure 10.1 **Ken Thompson (standing) and Dennis Ritchie, the inventors of Unix and C at Bell Laboratories, in front of a Digital Equipment corporation (DEC) PDP-11 minicomputer (early 1970s). The Association for Computing Machinery (ACM) awarded Thompson and Ritchie the Turing Award in 1983 for their work on operating systems theory.** (Courtesy of the Computer History Museum)

made several improvements to the design of the system, and he is responsible for building the first C compiler. The new tool was designed to be small and compactly described, but also highly efficient on emerging hardware systems such as the 16-bit DEC PDP-11 minicomputer. Between 1977 and 1979, the C language expanded further as its designers focused on implementation concerns, such as type safety and portability.[5] As the C language gained momentum, it was distributed widely with the Unix operating system, both inside and outside of Bell Labs. Every few years

5. Ritchie, "The development of the C programming language," 680.

a new release came along to support the growing platform. Of particular importance were the System III and System V releases from Bell Labs, and the Berkeley Software Distribution (BSD) releases created by a team working at the University of California, Berkeley. As Christopher Tozzi has explained in a recent book, BSD was especially important because it was a Unix clone that was eventually free of Bell Labs (later AT&T) source code—a stepping stone on the way to open-source system software.[6]

In 1978, Brian Kernighan and Dennis Ritchie published *The C Programming Language* with Prentice Hall, a primer that became the common reference point for C until an official American National Standards Institute (ANSI) standard was published in 1989.[7] *The C Programming Language* is an unusually important computer manual because it spread not only knowledge about the new language's syntax and design, but the text emerged as *the* teaching standard for how the language should be learned. Brian Kernighan's clear and concise prose was certainly an important reason for this. He devised effectual teaching conventions, style guidelines, and sample programs that many programming primer authors would adapt for their books. In addition, the guide book was co-authored by Dennis Ritchie, the creator of the C language, so the text had an aura of authenticity about it that few could match. Although Kernighan prepared the majority of the text, Ritchie collaborated on several sections, including the extensive reference that appeared in the last section of the book. The reference was essentially a blueprint for how the C programming language operated. Like many C programmers, I remember carrying around a dog-eared copy of my "K&R book" in the mid-1980s, and it has never left my bookshelf. Reading and referring to the primer was a common experience for many Computer Science students, self-taught programmers, and commercial developers of all types, from mainframe and minicomputer users, to solitary hackers, tinkers, and hobbyists. In January 2012, Prentice Hall announced that the second edition alone had gone through 49 printings. *The C Programming Language* became *the* perennial bestseller among computer programming primers.

As a sample of the teaching in *The C Programming Language*, consider the following C program designed by Kernighan and Ritchie to count the number of lines received as input in a terminal session. The complete program is just 10 lines long,

6. See Christopher Tozzi, *For Fun and Profit: A History of the Free and Open Source Software Revolution* (Cambridge, MA: The MIT Press, 2017), 20.

7. Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language* (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1978). The ANSI standard version was described in the updated edition, Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Second Edition (Upper Saddle River, NJ: Prentice-Hall, Inc., 1988).

including one blank line added for spacing and readability.[8] Note that the input lines are assumed to be terminated when a *newline* character (\n) is entered, which is typically appended to each line during the process of typing.

```
main()      /* count lines in input */
{
      int c, nl;

      nl = 0;
      while ((c = getchar()) != EOF)
            if (c == '\n\)
                  ++nl;
            printf("\%d\n", nl);
}
```

The program uses a *while* loop and an *if* statement to continuously evaluate the input received by the program. In 1978, the authors assumed that this input would come from a terminal attached to a DEC PDP-11 minicomputer (or equivalent), but there were other input options discussed later in the book. The *getchar()* function is used to examine each character in the input received. Nothing is done with the input other than testing whether the character is an end-of-file (EOF) marker or a newline character. (The first test is made in the *while* loop's conditional expression, and the second in the *if* statement that follows.)

When the EOF marker is encountered, the looping terminates, a sign that input from the terminal is complete and it is time to display the total number of lines received. Until this moment, however, the integer variable *nl* is incremented once for each newline character that the program encounters. The information is stored as a rolling total of the number of lines entered. The final *printf* statement in the program displays the value contained in the *nl* variable, so that the user can see the results of the program's work. It is a very simple demonstration program that appears early in the book, but typical of the Kernighan and Ritchie teaching method. The compact nature of the C language also lends itself to this exercise. Future technical writers were inspired by this step-by-step approach.

10.2 Learning C on Personal Computers

Enthusiasm for the C programming language grew rapidly. Although the first C programmers were minicomputers users, the language was designed for portability and soon implementations appeared on different machine architectures and operating systems. One of the earliest versions of C on a microcomputer was reported by a reader of *Dr. Dobb's Journal* in February 1979, soon after the

8. Kernighan and Ritchie, *The C Programming Language*, 17.

publication of Kernighan and Ritchie's *The C Programming Language*. Ted Shapin of Orange, California wrote in a "Letters" column that a "Tiny-C" product was available for $40 from a company in Holmdel, New Jersey, which could run as an interpreter in a computer fitted with an Intel 8080 microprocessor.[9] *Dr. Dobb's Journal* readers could purchase the printed assembly language instructions for the C-language "subset" via mail order. When the assembly language listing arrived, readers could type it into their microcomputer (provided there was a keyboard) and start writing C programs. Although limited in scope, this innovative Tiny-C interpreter could operate in just over 4KB of memory, making it suitable for the low-resource conditions of early PCs. A comprehensive manual was also included.

By 1980, several other "Small C" compilers had appeared in the marketplace from programmer-entrepreneurs Ron Cain, James Hendrix, and others. These rudimentary C products were all designed for resource-limited microcomputers and embedded systems, and they tried to approximate the C language specifications published in Kernighan and Ritchie's *The C Programming Language*. In some cases, the source code was simply released into the public domain for no fee. This free software approach followed the distribution paradigm advocated by several "Tiny BASIC" advocates in the mid-1970s. (For a history of Tiny BASIC, see Chapter 4.)

In October 1983, a regular C programming column appeared in *Dr. Dobb's Journal* entitled "C/Unix Programmer's Notebook." This feature was written by Anthony Skjellum, a talented software engineer involved with several innovative projects in the nascent PC software industry. Skjellum earned B.S., M.S., and Ph.D. degrees from the California Institute of Technology, and he has been a long-time member of the ACM. In 1984, Skjellum took up other work, and *Dr. Dobb's Journal* replaced his column with a recurring feature entitled "C Chest." This tips, techniques, and news column was written by Allen I. Holub, a talented technical writer who had studied Computer Science and Medieval European History at the University of California, Berkeley. Holub's popular "C Chest" column ran for 5 years in *Dr. Dobb's Journal*, contributing significantly to the diffusion of the C programming language in magazines. In the late 1980s and 1990s, Holub also wrote several reference books and primers on C/C++ programming, published by McGraw-Hill and other companies.

The movement behind C was quickly supported by magazine editors who sought to provide comparative content and product reviews for those considering C for professional use. In February 1985, *Computer Languages* magazine presented a "benchmarks" performance article that surveyed the industry's leading C compilers

9. Ted Shapin, "Review of Tiny-C owner's manual," *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia* 32 (February 1979): 41.

and judged how they competed on features, speed, and price.[10] The authors assessed the leading products on the MS-DOS, CP/M, CP/M 86, OS-9, and TRS-80 platforms.

Later that year, *Dr. Dobb's Journal* ran a similar benchmarks article comparing 13 C compilers designed solely for MS-DOS.[11] The list included Aztec C, Control C, C Systems C, Computer Innovations C86, Datalight C, DeSmet C, Digital Research C, EcoSoft C, Lattice C, Mark Williams C, Microsoft C, Software Toolworks C, and Wizard C. The presence of so many robust products indicated the vitality of the PC platform and the rapid emergence of C as a viable product for professional developers. In 1988 (the year the new ANSI standard was announced), a follow up essay in *Dr. Dobb's Journal* discussed how the various C products had changed over time and were being updated to promote the new standard. In this article, the Microsoft C Compiler was identified as the market leader, primarily because of Microsoft's unique position as a programming tool vendor *and* an arbiter of operating system standards.

Microsoft C version 5.1 was released in March 1988, and it offered integrated support for the Intel 80386 microprocessor, as well as the MS-DOS, Microsoft Windows, and OS/2 operating systems. (See Figure 10.2.) The product had a retail price of $450. Interestingly, Microsoft C 5.1 was only partially compliant with the emerging ANSI C standard. Partial compliance was typical of many of the commercial C compiler products released in 1988 and 1989 as the new language specification made its way into industry and gradually became the benchmark. Microsoft C version 5.1 was also a superset of Microsoft QuickC, a relatively recent "scaled down" version of the compiler that Microsoft had released for the novice and hobbyist markets. Released originally in November 1987, QuickC was priced at $99 to compete with Borland's Turbo C, which some industry analysts believed to be a superior product for non-professional audiences.[12] Both QuickC and Turbo C offered character-based integrated development environments (IDEs) and menu-driven help systems that were similar to the tools in Microsoft QuickBASIC version 4.5. QuickC version 2.0, released in January 1989, was fully compatible with Microsoft C 5.1.

The segmentation of the C programming marketplace into hobbyist and professional product categories indicates that the C language had successfully

10. Steve Leibson, Fred Pfahler, Jim Reed, and Jim Kyle, "Software reviews: Expert team analyzes 21 C compilers," *Computer Languages*, February 1985, 73–96.

11. "C Compilers for MS-DOS," *Dr. Dobb's Journal*, August 1985, 30–54. The review was conducted by the C Special Interest Group (C-SIG) of PicoNet and the Silicon Valley Computer Society under the direction of Richard Relph.

12. Richard Relph, David Chalmers, and Alex Khaloghli, "Product comparison: Six C compilers," *InfoWorld*, May 22, 1989, 47–60, here at 56.

Figure 10.2 **Microsoft C Compiler version 5.1 software disks (1988).** (Photo by Michael Halvorson; used with permission from Microsoft)

established itself on the IBM PC/MS-DOS platform. In less than a decade, C was thriving as a commercial product, and experienced software developers widely regarded the language for its structured programming elements, speed, and portability. Sensing an eager market for professional and new-to-topic learners, computer book publishers quickly responded with dozens of programming primers and reference books for the many users of C who needed help exploring the language and its application in real-world contexts. The following sections present the wide range of learning resources that were available.

10.3 Academic and Professional Resources

Although Kernighan and Ritchie's *The C Programming Language* served as the classic introduction to C programming, several attractive alternatives soon became

available. These new books were especially helpful for *true* novices, i.e., those with no previous exposure to programming concepts in any language. There were also C programming guides that emphasized a particular compiler or operating system, or which featured custom learning software to accompany the tutorial and its pedagogy. For example, Kernighan and Ritchie's text assumed a Unix system interface, but later books taught C in the context of popular PC platforms, including MS-DOS, Windows, and the Mac. The following sections introduce popular books in all of these categories.

We'll start with professional reference works. For C programmers who had already learned the basics, there were programmer's reference manuals and guide books that provided quick access to the reserved words and features of the language. One such text was *C, A Reference Manual*, by Samuel P. Harbison and Guy L. Steele, Jr.[13] This reference work was published by Prentice Hall, and it became popular in professional and academic settings. Harbison and Steele offered a complete definition of the C language, covering the major run-time libraries and an overview of the different versions of C. The second feature became useful as the language proliferated and went through revision by the International Standards Organization (ISO). This bestselling reference went through five editions, the last published in 2002.

Two well-known Math and Computer Science professors at the University of California, Santa Cruz also authored a popular C programming primer, published initially in 1984. Al Kelley and Ira Pohl's *A Book on C* was written in a lively style that had a structure similar to the classic Kernighan and Ritchie text.[14] The book introduced fundamental data types, flow control mechanisms, functions, branching statements, pointers, recursion, structures, list processing, and managing input and output in the Unix environment. The book was not organized as a concise manual to teach systems programmers (the approach implied by the K&R books). Rather, *A Book on C* introduced C as a general-purpose programming language that might have a variety of uses. Pedagogically, the Kelley and Pohl text was designed to be used in conjunction with a college Computer Science course, or perhaps as a stand-alone primer for experienced programmers. I personally encountered the book in a Microsoft-sponsored training course that taught Microsoft C in the context of IBM PCs and compatibles. Of particular value were the careful introductions to more complex topics like enumeration, unions, arrays, and self-referential structures—material that a student would appreciate when working with

13. Samuel P. Harbison and Guy L. Steele, Jr., *C, A Reference Manual* (Englewood Cliffs, NJ: Prentice Hall, 1984).

14. Al Kelley and Ira Pohl, *A Book on C: An Introduction to Programming in C* (Menlo Park, CA: The Benjamin/Cummings Publishing Company, 1984).

data structures and algorithms. The authors also valued simplicity. The book's first section began with two short sentences that encapsulate the convictions of many C programmers: "C is a small language. And small is beautiful in programming."[15]

Al Kelley and Ira Pohl were two well-connected academics with deep roots in 1960s computing. Al Kelley held a Ph.D. in Mathematics and joined the UC Santa Cruz Mathematics Department in 1966.[16] He became interested in computational mathematics at the university, and gradually helped to transform the department into one of the leading institutions for computational mathematics in the nation. ACM Fellow Ira Pohl received a Ph.D. in Computer Science from Stanford University in 1969, and he joined the Department of Computer Science and Engineering at UC Santa Cruz soon after. *A Book on C* was Kelley and Pohl's first publishing project together. The 1984 edition of the primer was designed for programmers using UC Berkeley 4.2 Unix and its popular C compiler.[17] An attribute of this system was their use of the venerable text editor *vi* to enter and edit programs. However, the 1990 edition of *A Book on C* expanded its treatment of the language by 100 pages and acknowledged other computing contexts. The book also included an appendix that documented the differences between traditional C and the new ANSI C standard.[18]

In later years, the third and fourth editions of *A Book on C* appeared, the last arriving in 2005 from the book publisher Pearson. Collectively, the series sold well, joining Kernighan and Ritchie as the most successful C programming primers in the academic/professional marketplace. Books in this category were sold primarily through university bookstores and professional organizations. As such, they had a longer shelf life than trade computer books, which were typically sold at retail outlets such as Barnes & Noble, B. Dalton, and Amazon.com. Soon Kelley and Pohl expanded on the initial success of their method, which they came to refer to as teaching "by dissection." This was a reference to their way of explaining programs via a structured walk through of the code, either section by section or line by line.[19] Eventually, Ira Pohl continued on his own, expanding his list of languages to include C++, Java, and C#. These primers included C++ *for C Programmers* (1989), *Object-Oriented Programming Using C++* (1993), *Java by Dissection: The*

15. Kelley and Pohl, *A Book on C*, 1.

16. Ralph Abraham, "UCSC Math: The Early Years," unpublished paper, August 16, 2017, 3. http://www.ralph-abraham.org/articles/MS%23153.UCSC/ms153.pdf. Accessed August 21, 2019.

17. Kelley and Pohl, *A Book on C*, 5.

18. Al Kelley and Ira Pohl, *A Book on C: An Introduction to Programming in C*, Second Edition (Redwood City, CA: Benjamin/Cummings Publishing Co., 1990).

19. For an example, see Al Kelley and Ira Pohl, *C by Dissection: The Essentials of C Programming* (Menlo Park, CA: Benjamin/Cummings Publishing Co., 1987).

Essentials of Java Programming (1999, with Charlie McDowell), and *C# by Dissection: The Essentials of C# Programming* (2002). For his commitment to computer science and programming instruction, Pohl was inducted as a Fellow of the ACM. His 2001 citation read, "For outstanding contributions to computer science research and education in the areas of heuristic search, analysis of algorithms, and programming language methodology." It was especially appropriate that his contributions to teaching programming languages was recognized, an undertaking he excelled at in print and in the classroom.

10.4 C Programming for the People

C was born in an engineering research lab and the language quickly gained admirers in academic and professional settings. However, there were also innovations in the trade press that allowed people with very different backgrounds to gain proficiency with the new programming tool. These books and learning systems were designed for self-taught programmers who had, at best, a limited exposure to coding practices through BASIC or a scripting protocol like DOS batch files. The following section reviews some of these creative products, which collectively helped to broaden the audience of programmers in America in the 1980s and 1990s. I'll note at the end of this section why I think C programming primers for popular audiences deserve more attention and represent a fascinating continuation of the learn-to-program movement in new contexts.

In 1984, The Waite Group created an important primer for self-taught programmers entitled *C Primer Plus*.[20] (See Figure 10.3.) This trade book was published by Howard W. Sams, a division of Macmillan that Mitchell Waite had partnered with on several occasions. (See Chapter 6 for a history of Waite's early programming books.) From the beginning, The Waite Group had demonstrated a skill for creating programming primers that would appeal to students, hobbyists, and self-taught programmers. This tutorial continued that trend. *C Primer Plus* was carefully designed and written by Waite, Stephen Prata, and Donald Martin—all coding experts with significant writing and teaching experience. Prata had earned a Ph.D. from the University of California, Berkeley and regularly taught Unix fundamentals and the C programming language. He worked at the College of Marin in Kentfield, California, where he had originally met Waite when he was a student. Donald Martin received an M.A. from San Jose State University and served in the mid-1980s as the Chair of Physics, Astronomy, and Energy Science at the College of Marin. Martin also taught Unix and C programming fundamentals to his students,

20. Mitchell Waite, Stephen Prata, and Donald Martin, *C Primer Plus* (Indianapolis, IN: Howard W. Sams & Co., 1984).

Figure 10.3 ***The Waite Group's C Primer Plus*, First Edition (1984), published by Sams Publishing. In its almost 30-year history, the six editions of *C Primer Plus* have sold over 550,000 copies, offering readers a hugely popular introduction to the C language.** (Courtesy of Mitchell Waite)

as well as Logo programming, the era's most deliberate attempt to create a language based on cognitive science. (For more on the origins of Logo, see Chapter 3.)

C Primer Plus found an eager readership and it became a strong seller, transforming the profile of The Waite Group and encouraging the small company to produce more programming titles for the expanding computer book market. The trade book market tended to move faster than the traditional academic or professional presses. For example, trade computer titles were typically designed, written, and published based on 18- to 24-month cycles that aligned with each new revision of a commercial software product. The majority of the sales for a new trade computer book would take place within the first 6 to 9 months after a book's release, which ideally coincided with the aggressive marketing campaigns initiated by the software publishers. Many of these releases were timed for early fall in the U.S. (September or October), so that both software and books could be fully in the retail sales channels before Thanksgiving and Christmas. However, academic presses were typically 2

to 3 times slower than this, managing their product development cycles around the rhythms of academic life, the peer review process, and corporate computing schedules. Rather than meteoric bestsellers, what the academic presses hoped for were "classic" programming titles that could remain in print and on the back list for years, such as Kernighan and Ritchie's *The C Programming Language* or Donald Knuth's *The Art of Computer Programming*.

In the 1980s and early 1990s, the major trade computer books publishers included Howard W. Sams, Microsoft Press, O'Reilly, Osborne McGraw-Hill, Que, Sybex, Wiley, Wrox, and Ziff Davis Press. The Waite Group established relationships with several of these firms, often arranging for "package deals" that allowed them to sign with one publisher for a number of computer books in advance. A significant advance on royalties would cover their start-up costs for each project, and, if successful, the books would pay The Waite Group additional royalties in the coming years. For example, in 1984 a New York publisher known as the New American Library wanted to get more deeply into computer book publishing, and they signed a 15-book deal with The Waite Group, accompanied by a $1 million advance on royalties.[21] In the 1990s, The Waite Group also experimented with authoring and publishing their own books, using Publisher's Group West (PGW) for distribution and sales. PGW, located in Berkeley, California, was one of the largest distributors of independent presses in the U.S.[22]

What made the Waite Group's *C Primer Plus* unique? Like the earlier C programming books that we have surveyed in this chapter, *C Primer Plus* included the essential material about working with data, using operators, managing program flow, creating functions, working with pointers, and gaining familiarity with the C run-time library. An important differentiator in the Waite Group's primer was the book's light tone and conversational feel. *C Primer Plus* featured humorous asides, entertaining illustrations, and practical advice. The text was supported by question-and-answer sections, programming exercises, reference materials, and contextual information about designing C programs for different platforms, such as MS-DOS and Unix. For example, the authors described how to program the ports of the 8088/8086 microprocessor and how to build functions that would play music on IBM PCs and compatibles.[23]

21. This book deal was arranged by an influential New York literary agent. I thank Mitchell Waite for providing the details of this and other business transactions in a series of interviews in June 2019. Most of the industry's multi-book deals were not so large, and there were considerable risks for both sides in such an arrangement.

22. In 2007, Publisher's Group West was acquired by Perseus Books Group.

23. See "IBM PC Music," Waite et al., *C Primer Plus*, 509–514.

The Waite Group soon followed up on their success, releasing new editions of *C Primer Plus*, as well as primers and references for other languages. One of the more pioneering projects they experimented with was *Master C* (1990), an innovative book-and-software package based on material from earlier titles and a creative software tutorial that they acquired from an interested reader.[24] The courseware allowed new C programmers to check their progress through software lessons on fundamental concepts. If they made mistakes, the *Master C* software showed them what their mistakes were and what they needed to fix. Although the software interface was rather primitive, *Master C* was one of several innovative book-and-software packages that attempted to teach new users programming concepts on the computer if they were unable or unwilling to attend local computer classes.[25] These products made their debut in the trade book markets years before the commercial Internet made online-learning a standard activity on platforms such as YouTube and Lynda.com.

At Microsoft Press, Augie Hansen's *Learn C Now* (1988) was the company's first book-and-software package that sought to teach new-to-topic programmers how to program in C using a combination of printed tutorials, software resources on disk, and a special edition of the Microsoft QuickC compiler.[26] This product sold for $39.95. It was a relatively good value because the book included a scaled-down version of the Microsoft QuickC compiler. (The scaled-down version was an otherwise fully-functioning product, but it could not create executable files and was essentially limited to running programs in memory.) The software came with a menu-driven, character-based IDE that was a significant improvement over using *vi* (the Unix text editor) to write programs. Augie Hansen was also a lively author who introduced C programming with thoughtful, informative prose, self-testing materials, and many sample programs that demonstrated the fundamentals of C programming. His *Proficient C: The Microsoft Guide to Intermediate and Advanced Programming* (1987) had established him as a leading author for experienced C programmers on the MS-DOS platform. *Learn C Now* focused his attention on the introductory market pursued by The Waite Group and others.

On the Macintosh platform, C programming advocates were also exploring the possibilities that C provided for working with the system's rich assortment of graphical user interface (GUI) features and resources. In 1991, Kurt Matthies and Thom

24. Mitchell Waite, Stephen Prata, Rex Woollard, *The Waite Group's Master C: PC-based Teaching System that Simulates a Real Instructor to Teach C Programming* (Mill Valley, CA: Waite Group Press, 1990).

25. For a review of The Waite Group's *Master C*, see Rick Ayre and Sue Ayre, "'Master C' tutorial offers solid fundamentals for C programming," *PC Magazine*, June 11, 1991, 67.

26. Augie Hansen, *Learn C Now* (Redmond, WA: Microsoft Press, 1988).

Hogan published *Macintosh C Programming by Example*, which used the Think C compiler to create interesting Mac applications.[27] Think C was an extension of ANSI C for the original "Classic" Mac operating system. The product was developed by THINK Technologies and appeared under the name Lightspeed C in its original 1986 release. Because Think C was essentially a subset of C++, the product also supported object-oriented programming, and it was conceptually similar to the architecture of Mac OS.

Matthies and Hogan welcomed traditional C programmers to the Macintosh platform, and they taught readers how to use important Mac components, such as MultiFinder, Dialog Manager, and QuickDraw. Although Think C gained many supporters in the early 1990s, the compiler could not successfully dislodge Objective-C, the programming language used in the NeXTSTEP operating system (NeXT computers) and later Macintosh systems.

As the C programming market became more sophisticated, there was a new danger—leaving the hobbyist or novice user far behind, with few pathways into the tantalizing world of commercial development. In the mid-1990s, Dan Gookin turned his attention to this important group of programmers, paving the way for popular coding initiatives in the years to come. I introduced Gookin in Chapter 6 as the author of *DOS for Dummies*, one of the bestselling computer books of all time for novice and experienced PC users. (See Figure 6.4.) With *C for Dummies* (1994), shown in Figure 10.4, Gookin used a similar approach to attract new-to-topic and hobbyist coders who felt intimidated or ignored by academic and professionally-oriented tutorials.

There were important reasons for people with non-technical backgrounds to be interested in the C language. First, C was gaining in importance in the U.S. software development community. By the late 1980s, an estimated 70% of commercial applications for the MS-DOS platform were written in the C language. Yet the tool continued to have a reputation for complexity and convolution, with a syntax that many found tortuous rather than elegant. A self-taught programmer himself, Dan Gookin reveled in this dynamic, and he approached the task of teaching C to newcomers with humor and a disarming attitude. This strategy began on the back cover of the book, which promoted the book and its target audience with humor:

> *C For Dummies*... is for you if you've tried to figure out C programming but have met with keyboard-pounding frustration. This book assumes you don't have a Ph.D. or work at MIT or Bell Labs. You're a bright person, but you require a bit more handholding than you have found in any other C book. You have

27. Kurt Matthies and Thom Hogan, *Macintosh C Programming by Example* (Redmond, WA: Microsoft Press, 1991).

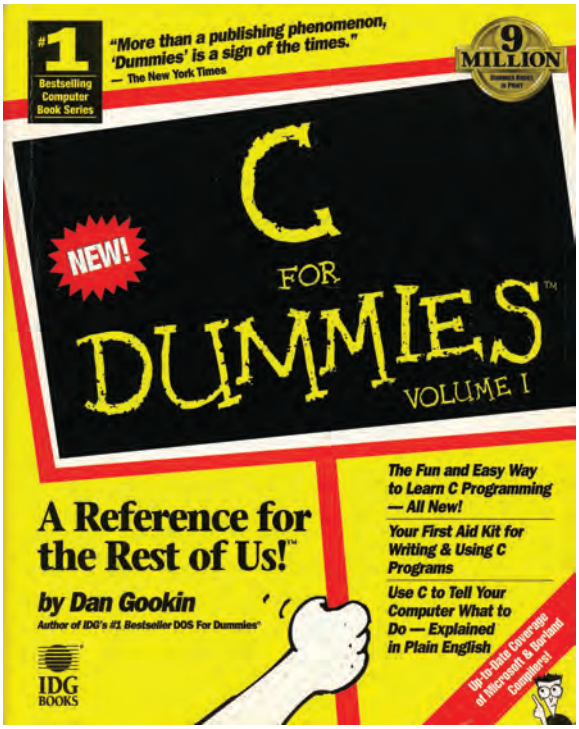


Figure 10.4 ***C For Dummies*** **(1994), by Dan Gookin, expanded the learn-to-program movement by welcoming hobbyists and self-taught programmers who felt marginalized by professional or academic approaches. Gookin's obvious love for programming made many converts.** (Cover image courtesy of Wiley Publishing and used with permission)

> your complier. You have the will. Fear not! This handy guide will get you up and running with the C language in an informative and entertaining way not yet conceived by any other programming book.[28]

The first pages of *C for Dummies* also welcomed users with conversational prose designed to close the gulf between the luminaries who created the language and the humble hobbyists who were experimenting with its syntax. Gookin begins:

> The guy who created the C programming language at Bell Labs is Dennis Ritchie. I mention this in case you're ever walking on the street and you happen to bump into Mr. Ritchie. In that case, you can say, "Hey, aren't you Dennis Ritchie, the guy who invented C?" And he'll say, "Why—why, yes I am." And you can say, "Cool."[29]

28. Dan Gookin, *C for Dummies*, Volume I (New York: Wiley Publishing, Inc., 1994), back cover.

29. Gookin, *C for Dummies*, Volume I, 8.

The book's chapters followed a similar approach, turning "serious" topics into fun diversions that intersected with popular culture. I list here part of the Table of Contents in *C for Dummies*:

> Chapter 1: The (Sometimes Painless) Beginner Stuff
>
> Chapter 2: Building (and Stumbling) Blocks of Basic C Programs
>
> Chapter 3: Weeping Bitterly Over Variables and (Gulp!) Math
>
> Chapter 4: Decision Making (or "I'll Have What She's Having")
>
> Chapter 5: Your Very Own Functions
>
> Chapter 6: Honing Your C Skills
>
> Chapter 7: Going Completely Loopy[30]

By not taking himself (or C) too seriously, Gookin ramped *down* the pressure on learning how to write code by giving readers a place to start. As Chris Bartocci of Rochester, New York, wrote: "I tried 3 other books on C and could not make anything out. *C for Dummies* has taken me a long way and now I understand a lot more."[31]

Why was this important? A more gradual approach was necessary as the learn-to-program movement threatened to run-aground on the rising complexity of building commercial applications for PCs. Software publishers like Borland and Microsoft were engaged in an arms race to equip commercial developers with better compilers and software development kits (SDKs). But not everyone was prepared to make the leap to commercial grade primers yet, or needed to. As this chapter has demonstrated, the PC platform had moved very quickly from Tiny-C interpreters to full-blown professional C compilers and SDKs—all in a matter of years.

Trade books like *C for Dummies* and *Master C* served as important entry points for students, power users, and tinkerers who aspired to be C programmers but were stumbling over the complexity of the existing materials. It is not hard to feel the spirit of computer literacy advocates like Bob Albrecht, Arthur Luehrmann, and David Ahl in Gookin's writing style and target audience. It was no longer necessary to make a specific case for *why* programming might be a worthwhile endeavor—there were those who doubted the value of learning to program in the early 1980s, but the commercial value of Pascal and C programming skills was obvious a decade later. Instead of appeals to the value of computational literacy, it was

30. Gookin, *C for Dummies*, Volume I, front matter.

31. Back cover quote, Dan Gookin, *C for Dummies*, Volume I and Volume II bundle (New York: Wiley Publishing, Inc., 1997).

important to provide a scaffolded approach to the coding skills that users wanted, because the process entailed rising levels of complexity and a number of different tools and skills.

The front cover of *C for Dummies* offers a subtle but important connection to the ideas and ethos of the learn-to-program movement. In the illustration, an outstretched arm holds up what appears to be a protest sign and shakes the sign in the air. Inside the sign are the words "C for Dummies." Next to the protest sign is a reading line for the book series: "A Reference for the Rest of Us!" The upper right corner of the cover also features a dial that indicates the number of Dummies books in print. In the edition pictured above (see Figure 10.4), the number exceeds 9 million copies. The overall effect is reminiscent of the illustration that the *People's Computer Company Newsletter* staff used when they depicted learning to program as a social movement. Beginning in 1972, they wrote, "BASIC is the people's language!" (See Figure 4.1.) Although Gookin's book and the Dummies series is more commercially oriented, the spirit of the learn-to-program movement continued in books like this, offering a distinct alternative to the academic and professional titles that commercial developers used, and which have received more attention. Even in commercial contexts, the grass-roots vision of the learn-to-program movement continued.

10.5 Charles Petzold's *Programming Windows*

The remainder of this chapter investigates another influential computer book from the trade press, Charles Petzold's *Programming Windows*, arguably the most popular training resource for C programmers learning to write commercial applications for the Microsoft Windows platform.[32] Like Peter Norton and Ray Duncan, Charles Petzold wrote numerous books for self-taught programmers who aspired to build professional-grade applications. The *Programming Windows* series became the best known of Petzold's efforts, but the reasons for this success were not simply the results of good writing and impressive market timing. The "Petzold books" also addressed an audience that was struggling more than most when it came to constructing non-trivial applications for PC-based platforms. Although Windows arguably made an IBM PC or compatible device easier to *use*, Windows applications were much harder for software developers to *build*.

32. The first edition is Charles Petzold, *Programming Windows: The Microsoft Guide To Programming for the MS-DOS Presentation Manager, Windows 2.0 and Windows/386* (Redmond, WA: Microsoft Press, 1988). The edition used for the examples in this chapter is Charles Petzold, *Programing Windows 3.1: The Microsoft Guide to Writing Applications for Windows 3.1*, Third Edition (Redmond, WA: Microsoft Press, 1992).

In terms of commercial opportunities, the market for Windows applications expanded dramatically after the release of Windows 3.0 in May, 1990. As I discussed in Chapter 6, this software release truly made Windows viable as a business and power user platform. However, there were numerous problems confronting developers who wanted to create commercial grade applications for Windows 3.0 and later. First, the existing SDK documentation was not adequate for software developers to make rapid headway into the product. Even if programmers had learned the basics of coding in C, they still needed to adapt their skills to the Windows operating environment, which required much more from developers than Unix or MS-DOS.[33] Moreover, there were not yet comprehensive development systems for IBM PCs and compatibles (such as Microsoft Visual Studio, which debuted in 1997). Early Windows programmers needed to use a wide range of tools and utilities to support the application development life cycle under the two most popular compilers, Microsoft C and Borland C.

Despite the formidable challenges, learning to write applications for Windows took on a kind of frenzied urgency for many programmer-entrepreneurs in the 1990s. Charles Petzold and his collaborators succeeded in meeting this need, and they were able to patiently explain a process that seemed daunting to so many. (See Figure 10.5 for three prominent Windows programming authors.) As the first editions of Petzold's *Programming Windows* were published, numerous industry observers lauded its value for teaching skills that were not easily transferrable. Peter Lewis of *The New York Times* endorsed the book by using the tantalizing imagery of a gold rush: "As one might expect, Windows programmers are in great demand these days, and this is the best book for programmers who want to cash in on the craze."[34] A reviewer in *Computer Language* magazine simply wrote, "Just take it as a given, if you're going to program for Windows, buy this book. It will pay for itself in a matter of hours."[35] Authorized and supported by Microsoft, *Programming Windows* became a fundamental resource for experienced C programmers who were ready to create commercial-grade GUI applications with either Microsoft C or Borland C. It was now possible to catch the wave.

Charles Petzold was born in New Brunswick, New Jersey, in 1953. He has lived most of his life on the East coast, residing primarily in New York City. In 1975, Petzold graduated with an M.S. degree in Mathematics from Stevens Institute

33. As of December 1991, there were approximately 80,000 C programmers using the Microsoft Windows SDK in the U.S.

34. Quoted in Peter H. Lewis, "The executive computer; who has really tried Windows?" *The New York Times*, December 2, 1990.

35. This *Computer Language* review is cited in the front matter of the Third Edition of *Programming Windows*; see Petzold, *Programing Windows 3.1*, FM.

Figure 10.5 **Charles Petzold (right) shown with fellow Windows programming authors Jeff Prosise (left) and Jeffrey Richter (middle), 2013. These authors wrote for *PC Magazine*, Microsoft Press, and other publishers, specializing in systems programming.** (Photo by Jeffrey Richter and used with his permission)

of Technology, a private research university in Hoboken, New Jersey. Although Petzold had some exposure to computers and programming through early work at the New York Life Insurance Company, he credits his interest in electronics to experiments that he made building musical instruments as a hobby, including a computer-controlled digital synthesizer.[36] These experiments contributed to his ability to write assembly language programs for early IBM Personal Computers, and he worked sporadically with PCs over the next few years. Gradually, Petzold's interests expanded to include operating systems and user-oriented topics. In January 1984, Petzold sent *PC Magazine* an unsolicited article about using the ANSI.SYS console device driver to modify the MS-DOS system prompt, one of the rudimentary attributes of PC systems that could be dramatically improved with tinkering and experimentation. *PC Magazine* sent him back a check for $800.[37] It was the first time that Petzold had been paid for his writing, and it eventually led to a career in essay writing and book publishing—first at *PC Magazine*, and

36. Charles Petzold, "Adventures in electronic music," Charlespetzold.com, September 2011. http://www.charlespetzold.com/etc/AdventuresInElectronicMusic/index.html. Accessed August 18, 2019.

37. Charles Petzold, "The Long (Essay) Version," Charlespetzold.com, July 4, 2008. http://www.charlespetzold.com/blog/2008/07/Software-Development-Meme.html. Accessed August 18, 2019.

later with *Microsoft Systems Journal*, *MSDN Magazine*, and Microsoft Press. Although Petzold wrote about MS-DOS, OS/2, Visual Basic, and C# programming, he became best known for his guides to Windows programming for C and C++ developers. His authoritative book, *Programming Windows* sold millions of copies and introduced experienced C programmers to best practices for creating applications for the emerging GUI standard on PCs.

Where did Microsoft Windows come from? Microsoft initially released Windows 1.0 on November 20, 1985. The graphical operating environment presented numerous opportunities—and some problems—for new users and software developers. First, the early versions of Windows were not technically operating systems at all, but graphical shells that ran on top of MS-DOS, requiring continuing familiarity with MS-DOS commands and procedures. From a user's point of view, Windows 1.0 was slow and cumbersome. It was challenging to switch among the "tiled" applications in a natural way, and unless you had a fast computer, such as an IBM PC AT or compatible. Unlike the Macintosh, a mouse or alternative pointing device was useful but not absolutely required for early Windows systems. This meant that some users continued to use "keyboard shortcuts" to perform common tasks such as opening menus, selecting commands, switching among applications, and drawing in a graphics program. Despite these quirks, the Windows software offered some advantages and improvements for traditional DOS users, such as built-in accessories, a clipboard for transferring information, and the ability to perform more than one task at a time (multitasking).

Windows was not the first GUI to run on a computer, of course. Earlier work on computer interfaces by luminaries such as Douglas Engelbart and the employees at Xerox PARC paved the way for numerous GUI experiments on microcomputers. Apple Computer introduced the Apple Lisa with its own version of a GUI in 1983 (see Figure 10.6), following by the Apple Macintosh, which used a similar layout, the following year.

Numerous systems debuted in the 1980s that employed a GUI framework for users to issue commands and run programs. The following list offers a selection of the systems that offered a GUI, along with the program's name and the date of the system's first use.[38]

- Unix and OpenVMS systems (X Window System, 1984)
- Sun Microsystems workstations (Network extensible Window System, mid-1980s)
- Commodore Amiga computers (Intuition/Workbench, 1985)

38. I have taken this summary of available GUIs from Petzold, *Programing Windows 3.1*, 5.

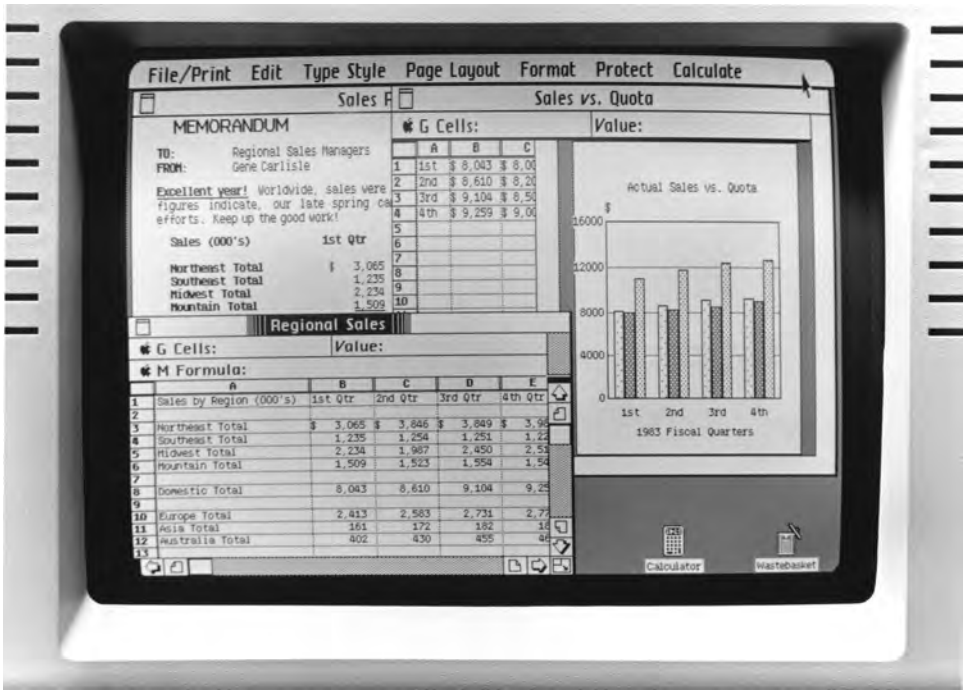


Figure 10.6 **The GUI of the Apple Lisa 2, shown with multiple applications loaded and visible on the screen. Depicted here (clockwise, left to right) are the programs LisaWrite, LisaGraph, and LisaCalc.** (Courtesy of the Computer History Museum. Used with permission of Apple Inc)

- Atari computers (Graphics Environment Manager, 1985)
- NeXT workstations (NeXTSTEP, 1989)

Although GUIs were easier to use in some respects than issuing text-based commands, users still needed training and orientation to the products. The first computer book for Microsoft Windows 1.0 users, *Windows: The Official Guide*, was published in May 1986 by Microsoft Press. This "how to" guide was authored by Seattle-area writer and entrepreneur Nancy Andrews, who owned and operated a business that provided technical documentation for companies in the Pacific Northwest. Andrews' text served as a practical, friendly introduction to the operating environment for new computer users and also experienced PC owners who had some prior experience with MS-DOS.[39] Andrews introduced common tasks,

39. Nancy Andrews, *Windows: The Official Guide to Microsoft's Operating Environment* (Bellevue, WA: Microsoft Press, 1986). Supporting Andrews' work as a technical reviewer was my first editorial job

discussed integrated report writing, and provided tutorials for the Windows Write and Windows Paint applications.

Microsoft Windows improved over the years, but the improvements added complexity to the operating system with the paradoxical result that as it became easier to use Windows it was harder to design and build applications for it. Charles Petzold captured this reality in his *Programming Windows* books with a statement that probably rang true for many who attempted to code for the graphical environment:

> Windows has the reputation of being easy for users but difficult for programmers. If you have no prior experience with programming for a graphical user interface, you should be warned right now that you will encounter some very strange concepts... If at first you find Windows programming to be difficult, awkward, bizarrely convoluted, and filled with alien concepts, rest assured that this is a normal reaction. You are not alone.[40]

The sentiment was so common that it became an inducement for lampooning Microsoft. It also caused some developers to return to the DOS, Unix, or Mac platforms.

So why was programming Windows so difficult?

Fundamentally, Windows was challenging because so many new expectations were being brought to PCs and PC software. Petzold summarized many of these issues in the introduction to the third edition of his book covering Windows 3.1, which I discuss below. (See Figure 10.7 for the book's front cover.).

- **The relationship between Windows and MS-DOS**. Windows did not fully replace MS-DOS, but ran on top of it until the release of Microsoft Windows Millennium Edition (ME) in 2000. Accordingly, new Windows programmers needed to know both the architecture of MS-DOS and the relevant structures and resources in several versions of Windows.[41]
- **Windows function calls**. Windows 3.1 supported over 1000 function calls to perform meaningful work, and the Windows programmer needed to learn a wide range of these. (The functions were collectively referred to as the *Windows API*.) In some ways, calling Windows functions and using related data structures was like using traditional C library functions, but there were many caveats and exceptions to learn.

---

at Microsoft Press. I started work on the project in late November 1985, learning the ropes from Andrews and colleagues Ron Lamb and Chris Kinata (née Matthews).

40. Petzold, *Programing Windows 3.1*, 10.

41. My overview of these features comes from *Programing Windows 3.1*, 11–15. I have contextualized them in a few places, such as the reference to Windows ME.

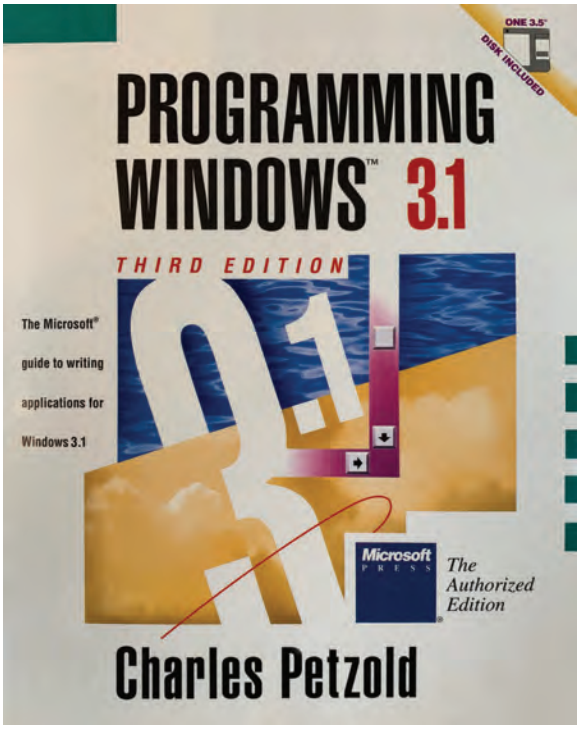


Figure 10.7 ***Programming Windows 3.1*, Third Edition, by Charles Petzold (1992). Widely praised in the popular press, *Programming Windows* was one of the most successful GUI programming primers on the PC platform.** (Used with permission from Microsoft)

- **Dynamic linking**. Windows executable files had the extension .EXE, like executable files under MS-DOS. However, the structure of a Windows application was different in several ways. One variation was that Windows executables required dynamic link libraries to work, which provided routines for loading programs, windowing commands, memory management, advanced graphics, and more. Rather than a simple C/C++ compiler, Windows developers needed a companion SDK to provide dynamic link libraries and the other support files that collectively comprised a Windows application.
- **Object-based programming**. An important skill for Windows programmers was learning how to work with objects of many types. Although Microsoft C was not initially an object-oriented language, the entire concept of Windows programming revolved around manipulating objects that had anthropomorphic characteristics and which could be replicated, modified, and shared. For example, a *window* in the user interface is a rectangular entity based on

the "windows class," which maintains visual attributes such as a title bar, menus, scroll bars, and other features. The attributes of a window could also be replicated, creating "child windows," which might have the same—or different—characteristics. Over time, both Microsoft and Borland added C++ support to their products, making object-oriented programming easier.

- **Message-driven architecture**. In the object-based programming environment of Windows, an "object" is a combination of code and data. For example, a window is an object, and there is code associated with the object stored in a special function known as the "window procedure." This procedure can send and receive messages from the operating system. Managing these messages, and the related "message queue," is an important task for the Windows developer.

Object-based programming required the Windows programmer to develop new conceptual models for thinking about constructing software. Rather than building an application that was largely in control of the computer and its resources, Windows programmers needed to develop programs that operated as "good citizens" in an ever-changing operating environment that supported many processes running at the same time. New issues such as exchanging information among programs, application security, controlling a range of devices and processors, and managing memory suddenly came to the forefront. The concept of *multitasking*, or allowing for the concurrent execution of multiple processes, also became important for Windows programmers to consider. In a multitasking system, the user (or the operating system itself) can interrupt a running program, save its state, load another task, and transfer control to it. Although Windows is able to manage some aspects of this behavior automatically, the programmer still must be aware of multitasking and plan for the possibility that their program will be interrupted many times before it runs to completion. In fact, a non-trivial Windows application might be made up of numerous sub-processes or *threads*, which run more or less independently and collectively carry out the work of the program.

A few more examples demonstrate Petzold's "divide and conquer" teaching method, which moved from the known world of C/C++ programming to the *terra incognita* of Windows development.

To get things rolling, Petzold respected tradition and listed a "Hello, Windows" program in the first chapter to show readers how the most basic Windows application might be created. This was analogous to the "Hello, world" program shown at the beginning of most programming primers, including Kernighan and Ritchie's *The C Programming Language*. The original K&R book begins with this routine:

```
main()
{
    printf("hello, world\n");
}
```
[42]

The authors carefully explained that the *printf* function sends the text "hello, world" to the screen (or attached teleprinter) when the program is compiled and run. The \n character indicates the end of a line of text and the beginning of a new line.

Charles Petzold's "Hello, Windows" program was different in many respects. While it did place the text "Hello, Windows!" in a window in the center of the user interface, it took over 80 lines of code to accomplish this work, as well as a build sequence that involved the Windows 3.1 SDK and either the Microsoft C/C++ Compiler version 7.0 or the Borland C++ Compiler version 3.1. To document what his 80-line program did in clear English, he spent over 30 pages of text explaining *why* such a long program was necessary and how it worked to display a basic greeting on the screen. He summarized the purpose of the added length with a single word, "overhead."[43] But he also described what readers would get for the effort: a complete Windows 3.1 application running in a rectangular window in the center of a sophisticated user interface. The new application would possess its own title bar, system menus, and resizing handles, the later visible so that the application's main window could be dynamically reshaped. The program could also be moved around the screen via mouse clicks and dragging motions, and it could be minimized, maximized, and closed (or terminated)—with familiar buttons and procedures, just like a commercial Windows application.

Petzold's proposition was that Windows programming wasn't so bad, once you understood what you were getting into. Each application contained the features that would allow it to function as part of a dynamic, multitasking operating environment. Naturally, there would be more to designing a Windows application than a simple "Hello, world" program in a text-based command shell. Configuring these features would become a source of intrigue and spirited conversation for developers that hoped to profit from the new system.

The heart of Petzold's sample program was a routine that created the "Hello, Windows!" greeting in the center of a new window. This routine is configured as a *message* named WM_PAINT in the system, which is sent to the application window each time that the window is opened, resized, restored, or uncovered. The process of displaying text on the screen is conceived of as "painting," but this is only one of many output options that can take place within an application window.

42. Kernighan and Ritchie, *The C Programming Language*, 6.

43. Petzold, *Programing Windows 3.1*, 15.

The steps involved in the painting process are more involved that simply sending output to a conventional character device. Instead, it is necessary to specify the window size, place the text in the vertical and horizontal center of the window, and then return control to the system. A fragment of the message logic Petzold devised looks like this:

```
switch (message)
    {
    case WM_PAINT :
         hdc = BeginPaint (hwnd, &ps) ;

         GetClientRect (hwnd, &rect) ;

         DrawText (hdc, "Hello, Windows!", -1, &rect,
                   DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;

         EndPaint (hwnd, &ps) ;
         return 0 ;[44]
```

Experienced C programmers will recognize the keywords *switch*, *case,* and *return* here, which establish a decision structure that tests an expression. Also familiar are the semicolon characters that end complete program statements. In addition to these familiar signposts, the parameter *hwnd* is a handle (or reference number) that refers to the new application's display window, and the parameter *&ps* is a pointer (an object containing the address of a memory location) to a structure that holds information that a window uses for painting.

An entity called a window *class* defines the general characteristics of a window, allowing the same window class to be used for creating a variety of different entities in the environment. Programmers create a window programmatically by calling the *CreateWindow* function, which identifies the window and specifies initial settings such as style, position, and size by way of parameters. A call to *CreateWindow* also return a handle to the created window which is saved in the variable *hwnd*. To manipulate the window again later, the programmer refers to this handle by name.

Here's what the call to *CreateWindow* looks like in Petzold's program code, which I provide here to show what readers encountered, and soon mastered, under his care:

```
hwnd = CreateWindow (szAppName,     // window class name
          "The Hello Program",      // window caption
          WS_OVERLAPPEDWINDOW,      // window style
          CW_USEDEFAULT,            // initial x position
```

44. Petzold, *Programing Windows 3.1*, 18.

```
        CW_USEDEFAULT,              // initial y position
        CW_USEDEFAULT,              // initial x size
        CW_USEDEFAULT,              // initial y size
        NULL,                       // parent window handle
        NULL,                       // window menu handle
        hInstance,                  // program instance handles
        NULL) ;                     // creation parameters
```

The 11 parameters listed here—the first 10 of which are followed by commas—are the essential attributes used to configure most application windows in a Windows 3.1 program. The Microsoft and Borland C compilers both recognized the // symbol as a marker for single-line comments, i.e., explanatory remarks recognized by the compilers but not translated into executable code. *szAppName* is the name of the *window class* for the program, an identifier that Petzold registered earlier in the "Hello, Windows" program. The *sz* prefix here is evidence of Hungarian notation, a variable-naming convention created in honor of the Xerox PARC and Microsoft software architect Charles Simonyi (1948–). To promote readability and self-identifying variables and objects, each name following this convention begins with a lowercase letter or letters that denotes the data type associated with the entity. In this case, the data type prefix in the class name *azAppName* is "sz", meaning "zero-terminated string." Hungarian notation was a cultural attribute of early Windows programming designed to support object-based development and reduce errors related to mismatched data types. However, the system has been employed less frequently in modern systems that display variable types via other methods.

The remaining parameters in the function call define the visible characteristics of a new window that is created. The default values for new windows are distinguished by using uppercase identifiers (WS_OVERLAPPEDWINDOW), and these represent numeric constants. In most cases, it would be too tedious for programmers to remember the actual numbers associated with all of these constants, so they are mapped to descriptive names in uppercase letters, many with two or three-letter prefixes to indicate how they are used. (For example, WS means "Window style.") These uppercase identifiers are defined in a file named WINDOWS.H, and they are documented in a manual called the *Windows Programmer's Reference* published by Microsoft. The identifiers are integrated with the HELLOWIN.C source code file during the compilation process via a *#include* statement at the top of the program listing.

10.6 On Complexity

This brief excursion into the content of Charles Petzold's "Hello, Windows" program has been included here to highlight the intricacies involved in creating applications for the Microsoft Windows version 3.1 operating system. By the early

1990s, GUI platforms like Windows were becoming much more sophisticated on PCs, reflecting the increased capabilities of hardware and software systems. It is interesting to contrast the *hardware complexity* of the first PCs with the *software complexity* of later PC platforms. In the early days of the Apple II and the first IBM PCs, the systems were so limited that programmers had to perform major programming feats in assembly language just to manage memory and allocate other system resources. It was the hardware that was a limiting factor, not the software. In later PC systems, however, the operating systems became so complex that the problem was not learning assembly language or high-level languages, it was managing the complexity of API-rich, object-based architectures that required a thorough knowledge of software development tools, graphical subsystems, multitasking, and message-driven architectures.

As an example of the mounting challenge, Microsoft Windows grew from a relatively modest operating environment in 1985 to a vast system comprised of tens of millions of lines of code by the mid-1990s.[45] Soon, it became virtually impossible for a single engineer to build a commercial Windows application on their own. It was necessary to work in teams, and to purchase a new raft of materials each time that the system was updated. In the realm of Windows programming, these titles included Jeffrey Richter's *Advanced Windows: The Developer's Guide to the WIN32 API for Windows NT 3.5 and Windows 95* (1995) and Jeff Prosise's *Programming Windows 95 with MFC: Create Programs for Windows Quickly with the Microsoft Foundation Class Library* (1996).[46] (For a photograph of the authors, see Figure 10.5.) Both books were designed to ease programmers through the difficult transition from 16-bit Windows programming to 32-bit Windows programming, which arrived in the era of Windows NT and Windows 95. The programming books were written by experienced author-entrepreneurs who, like Charles Petzold, had written for *PC Magazine*, Microsoft Press, and other technical publishers. Their references were long and complex, because 32-bit programming techniques made it possible to add mainframe-like features to systems that ran on ordinary Intel hardware. The complexity was daunting, and even Microsoft ran into trouble trying to manage the intricacies of these systems.[47]

45. Evans, Hagiu, and Schmalensee, *Invisible Engines: How Software Platforms Drive Innovation and Transform Industries*, 109.

46. Jeffrey Richter, *Advanced Windows: The Developer's Guide to the WIN32 API for Windows NT 3.5 and Windows 95* (Redmond, WA: Microsoft Press, 1995); Jeff Prosise, P*rogramming Windows 95 with MFC: Create Programs for Windows Quickly with the Microsoft Foundation Class Library* (Redmond, WA: Microsoft Press, 1996).

47. For an historical assessment of a few difficulties, see Stephanie Dick and Daniel Volmar, "DLL hell: Software dependencies, failure, and the maintenance of Microsoft Windows," *IEEE Annals of the History of Computing* 40, no. 4 (2018): 28–51, here at 32.

Video game programming under Windows presents a further case study in program complexity and the numerous challenges facing commercial application developers in the 1990s. For-profit video game development received a major boost in the mid-1990s with the introduction of several powerful home gaming systems, including the Sony PlayStation (1994) and Nintendo 64 (1996). Hoping to attract video game programmers to the Windows 95 platform, Microsoft introduced "DirectX" and the Windows Game SDK in 1995. DirectX allowed game programmers to access the rich multimedia features of Windows-based PCs and interact with protected memory. But this style of development could be frightfully complex, and new-to-topic developers often found the process bewildering, even if they already knew how to code in C or C++.

A pioneering author-programmer that rose to the challenge was André LaMothe, a Mathematics, Computer Science, and Electrical Engineering graduate of San Jose State University. LaMothe was a tinkerer at heart and largely self-taught in the genre of game programming. As a young man, he was an avid Dungeons and Dragons (D&D) player, and he was also inspired by *Tron* (1982), the groundbreaking science fiction film that combined live action scenes with computer animation.[48] LaMothe learned assembly language for the Zilog Z80 microprocessor, and experimented with graphics programming on the Atari 800, Amiga 500, and IBM PC AT. After several years of contract work as a software developer specializing in graphics, virtual reality, and artificial intelligence, LaMothe turned his attention to writing video game tutorials. His first books included *Tricks of the Game-Programming Gurus* (1994), *Teach Yourself Game Programming in 21 Days* (1994), *The Game Programming Starter Kit* (1995), *Black Art of 3D Game Programming* (1995), *Windows Game Programming for Dummies* (1998), and *Tricks of the Windows Game Programming Gurus* (1999). As evidence of the complexity of his subject, LaMothe's primers routinely included advanced mathematics, physics modeling, multithreaded programming, sound and multimedia tricks, and demonstration programs that approached commercial standards. Just to survey the required topics, many of LaMothe's books exceeded 1000 pages in length. Fearing that his tutorials would be too daunting and expensive for typical readers, more than one publisher asked LaMothe to reduce his page counts, for fear of losing sales.[49] But André was prescient, and by the late 1990s game programming had become a major topic of interest on the Windows platform. Without the video game development primers

48. I received this and other biographical information from Mr. LaMothe via email correspondence in June 2019.

49. Email correspondence, June 2019. Mitchell Waite, one of LaMothe's early collaborators, also supplied me with valuable information about early game programming on PCs.

that LaMothe and other programmer-authors prepared, the task would have been exceedingly difficult.

Admittedly, the complexities of GUI development often required additional training for aspiring developers. But the challenges should not take away from the accomplishments of the teachers who popularized this platform. As a measure of Charles Petzold's impact, consider that *Programming Windows* was published in six editions between 1988 and 2012. During this time span, the series developed into one of the best known book franchises of the PC era, selling millions of copies. As Microsoft Windows approached its 10 year anniversary, the software community acknowledged Petzold's contribution in guiding the learn-to-program movement through its GUI phase, a process that involved both commercial and pedagogical transformations. At COMDEX/Spring '94, Petzold was named one of seven "Windows Pioneers" responsible for the rapid expansion of the Windows platform. The luminaries selected were all identified as programming advocates who either led a major Windows project or supported the dissemination of information about the new platform.[50] The list of Windows pioneers included:

- Alan Cooper, the originator of Visual Basic, a rapid application development tool
- Lyle Griffin, the originator of Micrografx Designer, the first major graphics application for Windows
- Joe Guthridge, lead developer of the first Windows word processor, later sold as Lotus Word Pro
- Ted Johnson, lead developer for PageMaker, an early desktop publishing program
- Ian Koenig, lead developer of the Reuters Terminal financial program
- Ray Ozzie, the originator of Lotus Notes and chief technical officer at Microsoft
- Charles Petzold, the author of *Programming Windows* and other bestselling books

Like Peter Norton and Ray Duncan, Charles Petzold taught millions of programmers around the world the fundamentals of professional application development on PC-based systems. This intensive focus on programming *operating systems* would be just as important as the emphasis I placed on *languages* in earlier sections

50. For an interview with one of the recipients, conducted by ACM Fellow Wendy Kellogg, see "A conversation with Ray Ozzie: Cooperate, communicate, collaborate," in *ACM Queue* 3, no. 9 (December 16, 2005).

of this book. Both stages of the learn-to-program movement were important as technical knowledge about microcomputers and PCs was diffused throughout the U.S. From Tiny-C to Microsoft Windows, a collection of talented programmer-authors taught software development skills to hobbyists and aspiring professional developers. Those who learned to program in C joined ranks with the millions of developers who wrote code in assembly language, BASIC, Pascal, Forth, COBOL, FORTRAN, Logo, and other languages.

In the next chapter, I will examine the shift from individual programming tasks to team-based, commercial software agendas. I'll look at so-called "enterprise" development systems, product evangelism, industry trade shows, certification exams, and professional resources for self-taught coders—all technological "frames" that influenced how software development practices were gradually integrated into businesses and organizations in the U.S. We'll also learn how commercial development practices influenced the learn-to-program movement in the 1990s, changing its emphasis from computer literacy initiatives to the agendas of information technology (IT) professionals and the product cycles of the packaged software industry.

11 “Evangelism is sales done right”: PCs and Commercial Programming Culture

“This prestigious trade show – the largest of its kind in the United States – provides an ideal opportunity for consumers to examine the newest developments in computer systems, accessories, and services… Our success, as individuals and as nations, depends most on our ability to exchange ideas and to make the most of our knowledge and resources. That is why events such as the COMDEX trade show are so important.”

President George H. W. Bush, an open letter to COMDEX/Fall ’91 attendees, August 30, 1991[1]

This chapter presents an assessment of how commercial programming culture impacted the communities of Code Nation in the 1980s and 1990s. By “commercial programming culture,” I mean the business practices associated with designing, constructing, and marketing computer software, and the creation of commercial programming products for developers who advocated for the MS-DOS, Microsoft Windows, and Apple Macintosh platforms. As an episode in software history, this chapter is concerned with the transition from single-platform development tools for individual users (the mid-1980s), to the use of comprehensive “enterprise” development suites by the employees of Fortune 1000 corporations (the late 1990s). Professional and enterprise software suites offered sophisticated integrated development environments (IDEs), an impressive array of programming languages and compilers, platform-related libraries, project management systems, deployment tools, and extensive documentation. These all-inclusive products were advertised in lavish media announcements, computer magazines, and trade books, as well as

1. Excerpt from President Bush’s letter printed in *COMDEX/Fall ’91 Program & Exhibits Guide* (Las Vegas, NV: The Interface Group, 1991), 10.

fashionable events such as COMDEX, Macworld, Microsoft Developer Days, and TechEd.

I'll begin by examining Apple Computer's marketing strategies in the 1980s, most notably the ideas of Guy Kawasaki, an original member of the Macintosh marketing team. Kawasaki popularized the term *evangelism* to promote "vision-oriented" sales and marketing to consumers, and this strategy soon influenced the way that software products were designed, advertised, and sold throughout the personal computer (PC) industry. I'll also explore how PC products were marketed and presented at two important computer trade shows in the U.S. The first is the West Coast Computer Faire, a creation of Jim Warren and Bob Reiling in the 1970s. The Faire introduced microcomputers and related products to curious audiences in the San Francisco Bay Area, and it was emblematic of the less-formal, entrepreneurial conventions that here held in the early days of personal computing. At one of these trade shows, attendees might actually meet the developer of a new application or computer product, probably in a humble trade show booth decorated with pipe-and-drape walls and hand-made signs. (See Figure 11.1.)

COMDEX, a hybrid trade show for the national and international marketplace, launched what I call the *commercial computing trade show movement* in the 1980s, a progression of high-profile corporate trade shows that mixed members of the business community (company representatives and resellers) with technology journalists, industry luminaries, and members of the general public. At COMDEX, the corporate trade show booths evolved into large, multifunction workspaces where visitors could see new products demonstrated in theatrical environments supported by high-end multimedia systems. The methods and vitalities of American consumerism were increasingly on display at these spectacles, which could attract up to 200,000 attendees over the course of a week.

Finally, I'll discuss how computer programmers gained professional and enterprise-related development skills using new software, certification programs, and computer books in the 1990s. As a case study in the history of software commercialization, I'll focus on the ideas and technologies embedded in the Microsoft Visual Studio development system, first introduced in 1997.

When considered as a part of this book's argument about how and why people learned to write programs for PCs, this chapter investigates the impact of commercialization practices on early programming communities and the shifting objectives of the learn-to-program movement. By the early 1990s, programming had become a routinized professional business activity for thousands of software developers who earned a living by designing, implementing, testing, and maintaining software applications in the American PC industry. However, reports regularly surfaced that new projects were late, over budget, and filled with errors. These

Figure 11.1 **Lennie Libes (left) and Sol Libes in front of the S-100 Microsystems booth at the 1980 West Coast Computer Faire, held in San Francisco, California. Sol Libes was the founder of the Amateur Computer Group of New Jersey and creator of the Trenton Computerfest, one of the first microcomputer trade shows.** (Courtesy Jim Warren and the Computer History Museum)

problems must have felt like *déjà vu* for more experienced software engineers, especially those who cut their teeth during the mythological period of "crisis" in the mainframe computer industry. (For more on the crisis and the response of corporations and managers, see Chapter 2.) In the new era of corporate computing in America, the period of crisis was problematic, because the industry was governed by unpredictable consumer forces and changing corporate structures. The video game market collapse of 1983 was just one preview of the fickle nature of consumer demand, as was the infamous "dot-com crash" of 2000, when various online shopping companies went out of business. Poorly timed or buggy software could bring financial ruin to any software company, especially if they were clinging to an aging platform, inaccurate forecasts, or partnerships that were no longer strategic.

By the early 1990s, it was also clear that major new software releases would be planned and created *by large teams*. Gone were the days when a solitary programmer could build a breakthrough product on their own after holing up in a hotel room for a week with a clever idea. Commercial development projects were growing

exponentially in scope, and so were the challenges of testing and maintaining new software products. In 1992, for example, Microsoft Windows version 3.1 contained some 2.5 million lines of program code. The operating system was essentially a vast computing system created by numerous teams that worked in close cooperation for years. By 2001, the Microsoft Office application suite had expanded to some 25 million lines of code. Office was a comprehensive computing system distributed across a suite of programs, including Microsoft Word, Microsoft Excel, Microsoft PowerPoint, Microsoft Outlook, and Microsoft Access. In this new era, all the major commercial applications were designed, built, and tested by complex organizations that needed to carefully maintain their code bases, track feature requests and errors, plan for enhancements, and localize features for different language groups and cultures. As a result, companies like Apple Computer and Microsoft shifted their focus from the solitary hobbyists and developers that they had initially catered to, to the tools that would allow Fortune 1000 companies to manage their diverse enterprise and information technology (IT) needs.

How many people worked on the teams that created commercial computing systems? At Microsoft, the team that created the Windows NT operating system in the late 1980s started with 10 people. This core group created the underlying system (the Windows NT "executive") and the operating system's first subsystems. As the project grew, the Windows NT team expanded to 40–50 software designers and implementers, who added the next round of features and capabilities. Finally, the Windows NT team expanded to over 200 people to implement the final set of enhancements to the operating system. During this phase, team members built device drivers and tools, tested the system, and completed management, marketing, and support tasks.[2] The director who managed the development effort for the Windows NT 1.0 operating system at Microsoft was Dave Cutler, an experienced systems architect who had deep experience with managing development teams and adding to them over time. Cutler developed this expertise by designing several successful operating systems in the computer industry, including Digital Equipment Corporation (DEC)'s RSX-11M, VAXELN, and VMS (referred to now as OpenVMS). At Microsoft, he was known for bringing software engineering discipline to projects that were highly visible and influenced by internal pressures and marketing priorities.[3]

2. Helen Custer, *Inside Windows NT* (Redmond, WA: Microsoft Press, 1993), 13.

3. For Cutler's methods and a comparison of software development cultures at DEC and Microsoft, see "Oral history of Dave Cutler," interview by Grant Saviers, February 25, 2016, Computer History Museum, Fremont, California, 1–40, here at 21. https://archive.computerhistory.org/resources/access/text/2018/10/102717163-05-01-acc.pdf. Accessed July 12, 2019.

Of course, not all computer programmers were commercial or enterprise engineers in this important time of transition. The heyday of power users, hobbyists, gamers, and hackers continued in America's schools and rec rooms, and an assortment of part-time coders followed their individualistic ideas and agendas. But the impact of commercial software development practices would soon be felt broadly in the American economy, with social and financial consequences magnified well beyond the confines of Silicon Valley, Greater Boston, and the Pacific Northwest. Commercial software developers addressed the complexities of the software lifecycle with great enthusiasm and creativity, and many recognized the benefits that they could bring to American corporations. Their insistence on new methods and products increased the visibility of software engineers in America, and it also shifted the purpose of the learn-to-program movement from self-exploration, cognitive development, and solo entrepreneurship to building team-oriented development skills that would be of use to corporate America. The articulation of these ideas and enthusiasms began not in research labs or Computer Science departments but in the marketing divisions of Apple Computer and other consumer-oriented technology companies. These groups gradually developed new rationales for marketing and advertising the world's leading computer products.

11.1 The Macintosh Way

Guy Takeo Kawasaki was born in 1954 in Honolulu, Hawaii, and attended college at Stanford University, receiving a Bachelor's degree in Psychology in 1976. In 1977, Kawasaki enrolled at the Anderson School of Management at UCLA, where he earned an MBA degree. In 1983, Kawasaki was offered a position at Apple Computer, Inc., where he joined the teams that would market and sell the new Macintosh computer to customers. The original Mac was handsomely designed and meticulously produced, but it was also a new computing platform that had few available applications. If software was not created soon for the Mac, the intriguing product would fail, as had so many PC platforms before it. So, one of the first tasks of the Macintosh marketing team was to encourage software developers to build new applications for the computer and operating system. To be competitive, Apple needed business applications (word processors, spreadsheets, and databases), as well as games, productivity tools, communication programs, programming systems, and other core products. To communicate this message, they needed to describe the technical aspects of the system in compelling language, and they needed to inspire the belief that the Mac would be the graphical user interface (GUI) platform of the future.

While at Apple, Guy Kawasaki met a number of talented employees who were committed to the Macintosh project. For example, Kawasaki's former roommate at

Stanford, Mike Boich, was active in the Mac group, and he took the role of demonstrating the new system to any software developer that he could find. Kawasaki also worked with Mike Murray, a skilled marketing director at Apple. Murray had an MBA from Stanford, and he had worked on Apple teams since 1982. In the Macintosh group, he oversaw all advertising and public relations activities related to the new computer, including the famous "1984" TV commercial that ran during the Super Bowl on January 22, 1984. Murray also developed the concept of *software evangelism* that the company adopted, and he assigned various team members to act as software evangelists in the company—a new marketing role designed to inspire an almost-religious level of zeal and intensity for a product and its ultimate mission.[4] This term has a rather obvious connection to Protestant strains of Christianity in the Western tradition, as well as groups that have historically been associated with religious evangelism (i.e., *spreading the good news*). In more secular terms, the concept is sometimes described as "vision-oriented" sales and marketing.

The effort to market the new Macintosh to commercial software developers was one of the unheralded achievements of the mid-1980s, a time when commercial development tools were in a rudimentary state on most PCs. Moreover, many people associated Microsoft or IBM with marketing software development systems for PCs, not Apple. As Guy Kawasaki recalls, there was essentially no installed base for the new Macintosh computer. The device was limited to 128K of RAM, and the system had no hard disk. There was little in the way of product documentation, and a technical support system had not been established yet. Moreover, any development teams who joined Apple would be going up against IBM, a company that was ready to snuff them out.[5] Such a mission, according to Kawasaki, required sustained software evangelism. Beyond technical curiosity or even the motivation of making a profit, Mac application developers needed to be motivated to Do Things Right. These ideas were among the essential features of "The Macintosh Way," or what insiders called a marketing and customer relationship campaign that the company developed before and after the first release of the Macintosh in 1984.

In a later book, *The Macintosh Way* (1990), Kawasaki encapsulated the principles with a set of values about how software companies should care for customers. They included the following:

- Evangelism is sales done right. It is the sharing of your dream with the marketplace and the making of history with your customer. Evangelism is the purest form of sales. A Macintosh Way company doesn't sell; it evangelizes.

4. Kawasaki, *Macintosh Way*, 2.

5. Kawasaki, *Macintosh Way*, 2.

- Giving information and support to user groups is word-of mouth advertising done right. User groups are a medium like print or television, but you can't buy them. You have to earn them.
- Demos are sales presentations done right. Demos show customers why they should buy a product because they show how the product can increase their creativity and productivity.[6]

Given the later success of the Macintosh, Apple's core values quickly spread to other high-tech companies, especially to the thought leaders of PC software companies. When Mike Murray joined Microsoft in 1989 as Vice President of Human Resources and Administration, "The Macintosh Way" strongly influenced how Microsoft's product marketing groups moved forward. Before long, Microsoft had numerous software evangelists. These positions were often modified by adjectives that emphasized the product domain of the marketing promotion, such as "Visual Basic evangelist," "Windows evangelist," and so on. During my time at Microsoft, I routinely worked with product evangelists and often planned new books or software products with them. They were highly enthusiastic and often had very good ideas about marketing software.

In the context of this book, I highlight "The Macintosh Way" campaign as an important contributor to what I call the *commercialization of the PC book and software industry*, because product evangelism highlighted the economic and social benefits of a particular platform for developers and consumers. As other scholars have noted, it was primarily through *consistent platforms* that computer companies were able to build critical mass for a new technology and establish it as a technical standard in the marketplace.[7] Evangelization impacted programming culture because it infused software development norms with marketing strategies and the language of religious and moral conversion.[8] This mentality gradually became part of PC software culture in the 1990s, connecting software development practices with the psychological need for commercial success and personal fulfillment.

6. Kawasaki, *Macintosh Way*, 12.

7. On the importance of platform marketing, see David S. Evans, Andrei Hagiu, and Richard Schmalensee, *Invisible Engines: How Software Platforms Drive Innovation and Transform Industries* (Cambridge, MA: MIT Press, 2006), 97–101.

8. Modern scholarship on this issue begins with Max Weber, *The Protestant Ethic and the Spirit of Capitalism* (German: 1905/English: 1930). For a recent study that examines the connection between spiritual transformation and social movements (including the career of Steve Jobs), see Marion Goldman and Steven Pfaff, *The Spiritual Virtuoso: Personal Faith and Social Transformation* (London: Bloomsbury Press, 2017).

Enthusiastic marketing messages might also help to establish a company's brand identity, which connected a company's products to the needs, hopes, fears, and desires of American citizens. To get the word out, it was necessary to advertise in computer magazines, sponsor book projects, and develop partnerships with hardware and software resellers. Much of this work could be done at consumer trade shows, which allowed companies to exhibit their products, make deals with resellers, and find other potential partners. The next two sections discuss the West Coast Computer Faire and COMDEX, two innovative consumer trade shows that helped software publishers establish their brands and do the work of technology evangelization in the 1980s and 1990s. These events started out as regional conventions, but they soon attracted participants from around the globe. Each participant hungered to receive their own slice of the personal computing pie, and they found innovative ways to take it.

11.2 The West Coast Computer Faire

The West Coast Computer Faire was the brainchild of computing pioneers Jim Warren and Bob Reiling in the late 1970s. In the book *Hackers*, journalist Steven Levy describes how the two men decided to create, in good hacker spirit, an eclectic gathering that would gather together people who wanted to exchange information, equipment, technical knowledge, and "good vibrations."[9] Over time, the West Coast Computer Faire grew into this event. The Faire was styled after a regional "Renaissance Faire" in Marin County, California, which took place annually. As a technology showcase, however, the newly conceived "Computer Faire" would be open to the public and centered on the emerging world of personal computing.

We first met Jim Warren in Chapter 4. Warren was an early associate of Bob Albrecht who joined the staff of the People's Computer Company (PCC) to run *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia*, the first computer magazine to focus on PC software. Warren took that leadership position in 1975, but he also continued work part-time as an industry consultant. With Bob Reiling's help, Warren believed that he had enough time and contacts to launch a consumer trade show in the San Francisco Bay Area. The men's initial vision soon took the form of a convention and trade show, and they booked the San Francisco Civic Auditorium to accommodate as many attendees as possible.

On April 16th and 17th, 1977, more than 12,000 people visited the Civic Auditorium for the Faire. *Creative Computing* magazine reported that long lines were

9. Steven Levy, *Hackers: Heroes of the Computer Revolution*, Revised edition (Sebastopol, CA: O'Reilly, 2010), 222.

Figure 11.2 **The cover of *Creative Computing* magazine (July–August, 1977), announcing the results of the first West Coast Computer Faire.** (Reprinted with permission. © 2019 Ziff Davis, LLC. All Rights Reserved)

needed to accommodate the crowds. The most important reason for the crush was timing. (See Figure 11.2 for the cover of the *Creating Computing* issue that reported on the Faire.) In 1977, the country's first microcomputers were being announced, and representatives of many of the newest companies were present. For example, Steve Jobs and Steve Wozniak were both in attendance to discuss the features of the new Apple II computer. In all, there were over 180 exhibitors at the Faire, competing for customers and space to demonstrate their wares. The editor of *Creative Computing*, David H. Ahl, was on hand during the last afternoon to interview Jim Warren and review to what extent the event had been successful. (Ahl advocated for BASIC at the show, and he was regularly in the *Creative Computing* booth; as shown in Figure 5.1.) Warren's responses clarified what he understood as the *real* significance of the Faire: the trade show was essentially a continuation of the Free Speech Movement. It was a grass-roots campaign to bring computers to the people. "The point of doing this wasn't really making money," Warren said. "I mean, back in the 60's we had happenings in San Francisco, and San Francisco was meant for that,

and this is just another variation on it, except that it's a decade later. Back then, it was power to the people and now its computing power to the people."[10] The countercultural movement was gradually being transformed into one of the era's central computing mythologies.

Although the Faire began as a crusade for liberation, the trade show soon became a marketing and publicity opportunity that aggressive technology companies could not pass up. The region's major hardware manufacturers, software publishers, entrepreneurs, educators, and media representatives made a point of attending the trade show each year. This included authors and community members who wanted to teach classes, and publishers that wanted to sell programming primers, technical references, how-to books, and magazines. The annual event ran from 1979 to 1991, with crowds growing rapidly in the mid-1980s. Although most of the Faires took place in San Francisco, neighboring cities such as San Jose or Los Angeles were selected on occasion. To accommodate the many activities, Jim Warren's staff at the Faire grew to over a dozen people (see Figure 11.3), and they worked to arrange exhibiting space for corporations, teaching events with local experts, product demonstrations, lectures, and a festive banquet that allowed guests to network and strike deals. The list of attendees grew rapidly, and soon vendors arrived from out of state to participate. The Faire became the West coast's largest PC trade show, and it was typically held each spring. Figure 11.4 shows the crowds that flocked to the 1983 event. What is striking is the mix of businessmen wearing suits and ties, and the regular consumers dressed in casual clothing and swarming the trade show booths.

Did the West Coast Computer Faire encourage attendees to experiment with computer programming? The answer is "Yes" – especially during the early years of the PC Revolution, when commercial applications were hard to come by. The program guide for the 1981 West Coast Computer Faire held in San Francisco indicates that there were numerous programming classes on the topics of assembly language, BASIC, Forth, Pascal, and LISP programming. LeRoy Finkel and Jerald Brown (two associates of Warren and Albrecht at PCC), were listed as instructors offering tutorials on BASIC programming.[11] From the beginning, instruction in programming was a significant part of the Faire. In addition, there were always compiler and interpreter manufacturers present to represent their products. This group included well-established firms like Microsoft and smaller one- or two-person operations

10. David H. Ahl, "The First West Coast Computer Faire," *Creative Compu*ting 3, no. 4 (July–August, 1977): 99.

11. See *6th West Coast Computer Faire Program* (April 1981), accessed on March 28, 2019. https://archive.computerhistory.org/resources/access/text/2018/12/102725949-05-06-acc.pdf.

Figure 11.3 **The staff of the West Coast Computer Faire on a snowy day in April 1982. Jim Warren is wearing a brown coat and a wide-brimmed hat in the front of the group.** (Courtesy Jim Warren and the Computer History Museum)

that advertised in simple pipe-and-drape booths. In both cases, however, customers could often meet the actual developers of their products, who were eager to hear about how developers used their software and programming tools. Customers could ask for more information, register complaints, share sample programs, and request new features—all on the spot. The face-to-face experience represents a fascinating period of transition for personal computing and the learn-to-program movement. Gradually, microcomputer programming was transformed from a kit-based, swap-meet activity into a vibrant commercial industry, where product marketing and merchandizing activities were handled by trained professionals with corporate experience.

In 1983, Jim Warren sold the rights to the West Coast Computer Faire to book publisher Prentice Hall for $3 million. Later, Prentice Hall sold the trade show to American businessman Sheldon Adelson, who added the Faire to a portfolio of trade shows that he managed through the Interface Group Show Division. (The organization also included COMDEX, the annual trade show that took place each fall in Las Vegas.) Although the Faire's attendance gradually declined in the late 1980s, the event should be remembered as one of the pioneering consumer shows

Figure 11.4 **Crowds at the 8th West Coast Computer Faire, March 18–20, 1983.** (Courtesy Jim Warren and the Computer History Museum)

of the early microcomputer era. At the meeting, hardware manufacturers and software publishers met American technology consumers in person and sold their wares. In this context, companies were required to move beyond the traditional sales venues of mail-order catalogs and retail computer shops. They needed to create a compelling vision for customers and resellers that connected to the era's computing mythologies. The implied question was, "In what ways does your product bring computing power to the people?"

11.3 COMDEX and the Trade Show Movement

Consumer-oriented shows like the West Coast Computer Faire represented the communal, activist roots of the "PC Revolution." As the industry matured, these ideas were subsumed by what I call the *commercial computing trade show movement*, a steady progression of consumer-oriented trade shows that mixed members of the trade (computer company representatives and resellers) with consumers who wanted to learn more about new PC technologies for their schools, homes, and small businesses. This movement began in the mid-1980s, flourished in the 1990s, and gradually declined in the 2000s, when Internet-based commerce and other business transitions brought an end to oversized hybrid trade shows.

At commercial computing trade shows, new hardware and software products were announced, computer businesses established their brands, and consumers and resellers mixed to demo the newest products. Up to 200,000 or more gathered for these expositions in the U.S., and the event marked a high-point of the computer technology selling season. As many of the new products were designed for software developers (either emerging hardware/software platforms or new programming tools), commercial trade shows were also a key venue for the learn-to-program movement, as aspiring coders gathered to receive training, network, and experiment with the newest technologies.

Foremost among the commercial computing trade shows in the U.S. was COMDEX, a multi-day exposition that started in 1979 and held its last convention in 2003. The biggest COMDEX event was COMDEX/Fall, which took place each October or November in Las Vegas, Nevada. (Las Vegas was considered to be only venue large enough in the U.S. to hold the fall trade show and comfortably house and entertain all the attendees.) The COMDEX/Spring exposition typically rotated around the country, often in East Coast or mid-West locations. (For example, the first COMDEX/Spring venue was New York City in 1981.) Although I focus on the Fall COMDEX shows here, there were other computing trade shows that contributed to the transformation of the learn-to-program movement, including Macworld Expo, Windows World, and the Consumer Electronics Show. Outside the U.S., the most important expositions were CeBIT (Germany) and COMPUTEX Taipei (Taiwan). All of these venues represent important research opportunities for the historians of computing, business, and technology.

The Computer History Museum Archive in Fremont, California contains several useful folders of material related to the COMDEX shows, including a good selection of program and exhibits guides. These volumes often stretch to 600 pages or more, documenting the tremendous human effort and financial costs that it took to plan and produce these shows. (See Figure 11.5.) First and foremost, there were hundreds of trade show booths at the events, each staffed by employees or vendors who could explain the products or services offered by a company, present demonstrations, and clarify how products were distributed to value added resellers (VARs) or sold directly to customers. Some trade show booths were small, with humble pipe-and-drape fabric barriers that separated one company from the next, providing little more than a 6-foot table for products, a fabric back drop, and a power strip. (This was the spartan format of the West Coast Computer Faire.) But more common were the large "booth complexes" at COMDEX, essentially custom-made theaters and meeting spaces which provided a comprehensive experience for customers and visiting executives. These elaborate booths contained signage, office equipment, demonstration areas, and quiet spaces where executives could meet

Figure 11.5 **The COMDEX/Fall '90 Program and Exhibits Guide. This 600-page book filled with exhibits, events, and advertisements was given to each paying COMDEX attendee at the Las Vegas trade show.** (From the Michael Slater Collection on Microprocessors, COMDEX Program and Exhibits Guides; courtesy of the Computer History Museum)

peers. There was a large stage where up to 100 people might view a public demonstration. There were also smaller, more intimate demo booths where the individual details of product lines could be experimented with. Software companies paid great attention to the product demos, carefully rehearsing how the products in development might appear to the general public. On occasion, both employees and customers were forced to encounter software bugs, system crashes, and the "public rebooting" of fragile systems. (For IBM PCs and compatibles, this was accomplished via the infamous key combination Ctrl+Alt+Del.) For these reasons, product demos were usually handled by senior engineers, trained product managers, and (in some cases) well-groomed professional actors wearing headsets and bright shirts.

Company employees at the trade show booths dressed "casual corporate" in the late 1980s and early 1990s, the men with khaki pants and corporate-logo polo shirts, and women with more formal attire, typically blazers, white shirts, A-line

skirts, nylon stockings, and pumps. "Women dressed more formally than the men in the trade show booths," remembers Kim Halvorson, a Microsoft employee in U.S. Sales and Marketing in the late 1980s and early 1990s. "Wearing nylon stockings was warm and they always ran; we needed many pairs in our arsenals. It was also the era of shoulder pads and power jewelry, which women often wore in professional settings to project an aura of strength."[12] In other booths, there was an effort to wear matching shirts, company buttons, scarves, and other accessories that gave employees a uniform look. This emphasis on clothing made COMDEX feel a bit more formal than the casual "street attire" of the West Coast Computer Faire, but the commercial computing trade show was still less formal than the starched suit-and-tie guises of IBM sales and marketing personnel in the 1960s and 1970s. The overall focus was to impress commercial customers and resellers, not executives.

The COMDEX/Fall '88 Program and Exhibits Guide indicated that there were over 1,700 exhibitors displaying their wares in trade show booths spread out among several Las Vegas hotels and convention centers.[13] The guide contained a schedule of events and several maps to help visitors locate the product booths and teaching/information sessions. The show took place over 5 days between November 14th and November 18th, 1988. As tens of thousands of attendees moved from one booth to the next, they carried plastic bags to hold the buttons, flyers, and "swag" (free promotional items) distributed by company representatives. To attract the attention of customers, many of the buttons displayed humorous slogans or phrases, such as the button in Figure 11.6, which contains a pun related to programming and music from the Beatles. ("We all live in a yellow subroutine.")

In the era before the commercial Internet, free software and demonstration programs were sometimes distributed at trade shows via floppy disks or CD-ROMs. By the mid-1980s, software vendors realized that it was too much work to type in source code manually, which had been a mainstay of early microcomputing. But it was hard to locate complementary copies of commercial-grade software, such as Aldus PageMaker, Lotus 1-2-3, or Microsoft Word. Loading free trade show software on your home or work computer was also a security risk by the late 1980s—it could be the source of *malware*, or a destructive computer virus. To combat any such risks, high-profile corporate customers were sometimes given clean, shrink-wrapped copies of commercial software at COMDEX, which they added to their bags of tee-shirts and corporate swag.

12. Kim Halvorson, unpublished interview with the author, March 31, 2019. The clothing reference is to COMDEX/Fall '90, which Kim attended with the Microsoft group.

13. *COMDEX/Fall '88 Program & Exhibits Guide* (Las Vegas, NV: The Interface Group, 1988).

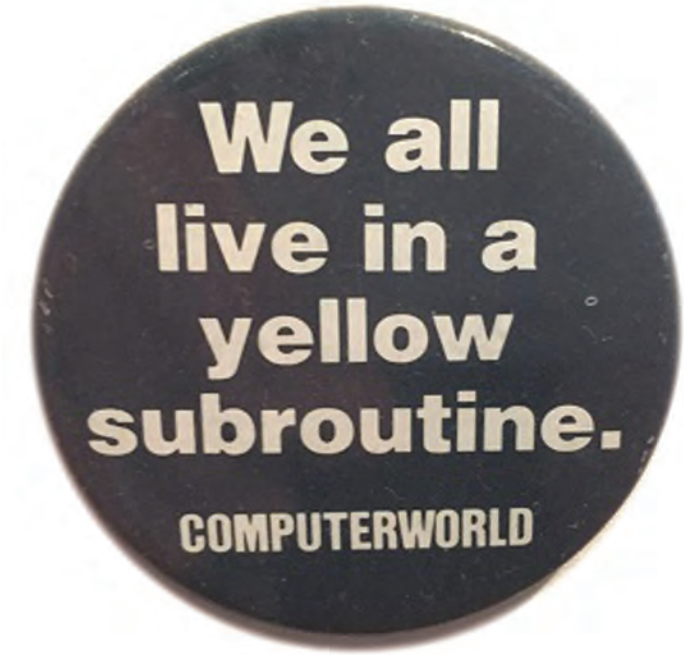


Figure 11.6 **Computerworld magazine button distributed in the 1980s. "We all live in a yellow subroutine" was a reference to the feel-good nature of Beatles music and the collaborative spirit of the learn-to-program movement.** (Photo by Michael Halvorson. Courtesy of the Computer History Museum)

The COMDEX/Fall '90 Program and Exhibits Guide indicates that the November 12th to 16th trade show had expanded to include exhibits and activities in the Sands Expo and Convention Center, the Las Vegas Convention Center and West Hall, Bally's Casino Resort, Caesars Palace, the Las Vegas Hilton, and the Riviera Hotel.[14] The COMDEX/Fall '90 show cost attendees $295 for the full 5-day experience, or about $150 per day. For those consumers who simply wanted to visit the exhibitor's areas, where all the products were visible, they could pay $75 for all 5 days. The guide encouraged attendees "to wear their badge at all times," a directive that brought humorous consequences in the evenings, when many attendees continued to wear their lanyards around Las Vegas—in the bars, in the all-you-can-eat buffets, and even while placing bets at some of the gambling tables. I witnessed this first hand while attending COMDEX/Fall '90 as an Acquisitions Editor for

14. *COMDEX/Fall '90 Program & Exhibits Guide* (Las Vegas, NV: The Interface Group, 1990).

Microsoft Press, exploring the product booths, identifying trends for potential projects, and working to meet new computer book authors. I wore a Microsoft Press logoed polo shirt for many of my shifts at the Microsoft Press exhibitor's booth. In the evenings, it was customary for me to wear suits at the formal dinners with clients and at the cocktail parties with colorful industry personalities. This is where, at least once a year, I would shake hands with Dan Gookin, Kris Jamsa, Steve Nelson, Charles Petzold, Neil Salkind, Craig Stinson, Mitch Waite, and other authors in the Microsoft Press stable who regularly attended the show. Dean Holmes, Senior Acquisitions Editor at Microsoft Press, described these meetings as useful cultivation sessions that were hard to replicate in other ways. "At COMDEX, we could connect with current authors and future prospects, trade stories, and gather information about upcoming products and their ever-changing release schedules. It was an opportunity to thank our authors, have a beer, and put out fires when problems occurred."[15]

Although business networking was an important aspect of COMDEX, it was also useful to hear the technology predictions from industry experts, and to attend breakout sessions where specific platforms or products were discussed. Each COMDEX show had a high profile keynote address, where thousands of attendees gathered to hear from an industry luminary. These events were held in the biggest ballrooms, where there was a formal stage, theatrical lighting, and an impressive A/V system. At COMDEX/Fall '90, Bill Gates was the keynote speaker, and the exciting news was Windows 3.0, which had been enhanced with virtual memory, loadable device drivers, and a new look. Over the coming days, there were conference panels and roundtable sessions with many of the industry's best-known figures, including Stewart Alsop II (*PC Letter*), David Archambault (Commodore), Steve Ballmer (Microsoft), Dan Bricklin (Slate), Esther Dyson (*Release 1.0*), Gordon Eubanks (Symantec), Bill Machrone (*PC Magazine*), Tom F. Wheeler (IBM/General Electric), Amy D. Wohl (Wohl Associates), and Will Zachmann (*PC Magazine*).

As the list indicates, most of the featured speakers were either industry executives, publishers, or technology journalists. For a time, magazine editors and computer book publishers were among the most influential groups at COMDEX, because they promoted the industry's hardware and software products to millions of customers each year via newsletters, books, and periodicals. As evidence of this trend, the COMDEX/Fall '90 guide announced that 27 computer book publishers were present at the show and exhibiting, each with a significant staff of marketing and sales personnel, as well as acquisitions editors to sign up new projects. All the major PC hardware and software companies were also visible,

15. Dean S. Holmes, email correspondence with the author, September 21, 2019.

including Apple Computer, Autodesk, Borland, Digital Equipment Corporation, Hewlett Packard, IBM, Intel, Microsoft, and Sun Microsystems—just to name a few. Most of these companies were aggressively pursuing new deals, partnerships, and co-marketing arrangements. For example, the "Intel Inside" co-marketing campaign was launched about this time to raise the profile of the Intel microprocessors that were inside many of the industry's PCs. The only groups that were *not* present at COMDEX were the traditional mainframe, supercomputer, and minicomputer vendors that did not sell products directly to retail customers.

How many exhibitors, consumers, and resellers attended these shows? The COMDEX/Fall '91 Program and Exhibits Guide included a statement from Interface Group President Jason E. Chudnofsky answering this question. Since its founding 12 years earlier, Chudnofsky claimed that some 750,000 paid computer industry attendees had attended COMDEX.[16] Seeking to benefit from the recent upswing, The Interface Group had raised the price of admission to $450 for the 5 day trade show and exposition. This was a 90% increase from the previous year. There was also no option for an "exhibition only" ticket. The comprehensive fee was now the standard price for anyone who wanted to attend the multi-day event. But the industry heavyweights kept coming. In 1991, the keynote speaker was Andrew S. Grove, Ph.D., of Intel Corporation. There was also a well-publicized CEO roundtable event featuring Bill Gates (Chairman and CEO, Microsoft), Philipp Kahn (Chairman, CEO, and President, Borland International), and Jack D. Kuehler (President and board member, IBM Corporation).

By the early 1990s, over 140,000 people were attending COMDEX/Fall each year, and the numbers would increase steadily until 1996, when attendance peaked at 225,000.[17] By this time, the growth of the industry sector was undeniable—personal computing had become a major contributor to the American economy, and commercial computing trade shows served as conspicuous showcases of American capitalism and technology. Hardware and software manufacturers were spending millions of dollars marketing their products and cultivating brand loyalty. As personal computing gained momentum, software developers became a valued part of this world.

One of the important artifacts of this era is a fascinating open letter written by President George H. W. Bush to COMDEX/Fall '91 attendees, placed prominently on the first page of the Program and Exhibits Guide. In the letter, President Bush welcomes visitors from around the world to Las Vegas, and he encourages them

16. *COMDEX/Fall '91 Program & Exhibits Guide*, 622.

17. Ted Smalley Bowen and Carolyn A. April, "Comdex starts to lose some luster," *Info World*, November 25, 1996, 3.

to participate fully in the glittering world of consumer technology that America has showcased. To an historian with training in the pageantry and pomp of earlier eras and leaders, the act seem reminiscent of the greeting that Queen Victoria offered to international visitors at the beginning of the Great Exhibition in London (1851), an important showcase of the late industrial revolution. President Bush greets COMDEX attendees in this way:

> I am delighted to extend warm greetings to everyone who is attending the COMDEX/Fall '91 Conference and Exposition. My special greetings to our visitors from abroad.
> This prestigious trade show – the largest of its kind in the United States – provides an ideal opportunity for consumers to examine the newest developments in computer systems, accessories, and services. In this rapidly changing, increasingly technological world, it is vital that computer users stay abreast of the many different products and programs that are available to them. Our success, as individuals and as nations, depends on our ability to exchange ideas and to make the most of our knowledge and resources.
> That is why events such as the COMDEX trade show are so important.
> Barbara joins me in sending best wishes for a productive event and for every future success.
> George Bush[18]

Let me add a bit of context to this. In the 1980s and 1990s, it was common for American presidents to welcome businessmen and international visitors from abroad. In fact, the organizers of the COMDEX trade show had arranged for several presidential letters in exposition programs, including similar greetings from President Ronald Reagan. However, Bush's letter was the first to be accompanied by a photograph depicting a sitting president using a PC in the Oval Office. In the image, President Bush sits typing with two hands on the keyboard of what appears to be a high-powered IBM PC or compatible computer, with a video graphics array (VGA) monitor and a personal laser printer nearby. The link between the American economy, global commerce, and personal computing never seemed so striking.

11.4 The Trouble with Self-taught Programmers

COMDEX finally closed its operations in 2003, after a period of consolidation and decline in the business of commercial trade shows. By that time, the "dot-com" bubble had burst and the early enthusiasm for Internet-related businesses resulted in retrenchment for the computer industry. Apple Computer, IBM, and

18. *COMDEX/Fall '91 Program & Exhibits Guide*, 10.

other technology companies abandoned COMDEX for smaller corporate events and product showcases. Consumers continued to shop for and buy new computers and software products, but they learned about them in new ways.

What did this mean for the learn-to-program movement in the U.S.? Without major trade shows, how did programmers learn about new software development products? Was the number of active coders rising or falling in the U.S.?

In the 2000 U.S. Census, there were 521,105 full-time year-round workers who identified as "Computer Programmer" in their job title. An additional 595,965 people selected "Computer Software Engineer" as their occupation. Adding these numbers together, we get 1,117,070 people who self-identified as residents working in programming-related jobs. In addition, there were 554,720 who selected either "Computer Scientist" or "System Analyst" in the 2000 U.S. Census. This brings the total number employed with professional programming skills in the U.S. to 1,671,790 in the year 2000.[19] Compared to earlier statistics, this indicates a steady rise in the number of professional programmers in the country.

As I have emphasized in earlier chapters, many of these professional programmers were self-taught, and they used books, magazines, and self-paced training courses to cultivate their skills. In addition, there were millions of non-professional U.S. residents who had learned to write code in some form but did not account this skill as their primary occupation. These were likely scientists, information workers, hackers, and "power users" who could use programming tools at work. These were likely scripting systems, HyperCard stacks, AWK programs, and language compilers associated with FORTRAN, Forth, QuickBASIC, Turbo Pascal, C/C++, Java, and other products.

Curiously, the non-professional group gained some notoriety in the 1990s. By this time, it was commonly understood that millions of Americans knew something about computer programming and that they could use this skill for good or ill. Within the corporate IT community, there were some who believed that amateur programmers might somehow threaten the computing infrastructure if they were allowed to write programs that were widely distributed. It was an anxiety related to the fears about hacking in the 1970s and 1980s, which threatened to disrupt telecommunications systems across the country. IT executives mused: how safe are our organizations now that so many people can build applications with little or no supervision? Are the new rapid application development tools making software development too easy? Could the grassroots learn-to-program movement actually lead to problems with American's high-tech infrastructure?

19. 2000 U.S. Census data summarized in Evans, Hagiu, and Schmalensee, *Invisible Engines*, 84.

As odd as this line of inquiry sounds, the COMDEX/Fall '90 trade show had at least one panel exploring the dangers of amateur coders. The session was entitled "The Challenge of Personal Programming," and it featured a panel of IT experts who readily expressed concerns about how easy it had become to write programs in America.[20] Might this newfound freedom encourage "unqualified end users" to write code that was dangerous or of poor quality? Should the IT community band together to stop "personal programming" before it created too much trouble? Would the publishers of programming tools and compilers be liable or complicit if their technologies were used for ill and not good?

The chair of the COMDEX panel was Michael Edelhart, Editor-in-Chief of *PC/Computing*, a Ziff Davis publication. Contributors included Bruce Barrington (Clarion Software), Fred Gray (Microsoft), Frank King (Lotus Development Corporation), and Eugene Wang (Borland International). Of special concern to the panelists were the industry's new menu-driven IDEs that made it so easy for "non-technical users" to write programs.[21] While the Microsoft and Borland panelists defended their company's programming systems, all present admitted that *better training resources* were necessary to ensure the safety and reliability of new systems, which could quickly integrate themselves into the business activities of America's corporations. In particular, a few IT and management information system (MIS) managers believed it was high time to exert *system-level control* over uncooperative users and their code. In short, some sensed danger in the learn-to-program movement, and they worried that America's computing infrastructure might soon be threatened by amateur coders with dubious training.

Was the concern that self-taught hobbyists were somehow embryonic hackers who were eager to disrupt America's technical systems? Or was the concern that self-taught programmers were likely to end up in professional jobs, lowering institutional standards in some way?

In the June 1991 issue of *Dr. Dobb's Journal*, Andy Bender from Maryland supported the second concern. Echoing a familiar complaint from the 1960s, Bender worried that there were few engineering practices discernible in the PC industry. He laid some of the blame on "learned-it-on-the-job programmers" (the *italic* emphasis is mine):

> Dear DDJ,
>
> ...
>
> *I'm getting really frustrated with many aspects of the software development business*, especially in the world of business applications: incompetent managers

20. "Conference Session," *COMDEX/Fall '90 Program & Exhibits Guide*, unnumbered page.

21. "Conference Session," *COMDEX/Fall '90 Program & Exhibits Guide*, unnumbered page.

> placed in charge by senior executives who know little (or less) about the software development life cycle (Senior Exec: "I'll put Paul in charge; he's an accountant, but he did something on the installation of our general ledger package, so he must know all about computers…"); incompetent, learned-it-on-the-job programmers ("structured what? I've been in this business for twenty years. Nobody can teach me anything about programming…"); absurd project schedules ("Complete specifications before you start coding? No way, there isn't time. Start coding now or you won't make your deadline…"); etc.
> *What will it take before software engineering is considered a real profession, requiring completion of a standard university curriculum and subsequent licensure, before putting code in a buffer for money?* …
> DDJ is a respected magazine, a voice that is heeded by programmers. I hope to hear it much louder in favor of *professional software engineering standards* whenever and wherever discussions of this onerous problem occur.
> Andy P. Bender
> Riverdale, Maryland[22]

Bender's complaint was that software developers were being controlled by executives who knew little about software engineering principles and the dynamics of the product development life cycle. Moreover, as earlier generations had done, the letter writer laments the lack of professional standards, including the completion of degree programs and/or training courses where best practices are introduced.

The concerns about casual programmers attacking America's infrastructure were clearly overblown. But as the PC software industry reached new levels of sophistication, were there new conduits through which professional development practices might flow? Was there a way that academic research might influence the learn-to-program movement as it entered more commercial contexts?

11.5 Software Engineering for the People

In *Code Complete: A Practical Handbook of Software Construction* (1993), professional software developer Steve McConnell considered the same problem.[23] His best-selling "crossover" book was one serious attempt at addressing an issue that had become a stumbling block for the commercial software industry—how could professional engineering practices be dispersed more effectively to self-taught programmers who were taking development jobs at leading software companies? How could academic research in computer science support the new commercial

22. Andy P. Bender, "Letters," *Dr. Dobb's Journal*, June, 1991, 12.

23. Steve McConnell, *Code Complete: A Practical Handbook of Software Construction* (Redmond, WA: Microsoft Press, 1993).

programmers who were designing, constructing, and maintaining the country's widely used systems?

McConnell had professional experience writing more than 50,000 lines of production code over a 5-year period for software companies in Washington State, including Microsoft Corporation. He received a B.A. in Philosophy from Whitman College and an M.S.E. in Software Engineering from Seattle University (1991). These experiences provided him with a grounding in the liberal arts and an important connection to the contemporary discipline of Computer Science, which had developed a significant research base by the late 1980s. McConnell was particularly interested in what he described as the "gap between the knowledge of industry gurus and professors, on the one hand, and common commercial practice on the other."[24] He argued that approximately 100,000 new computer programmers entered the profession each year, but only about 40,000 Computer Science degrees were awarded, creating an obvious gap in the formal instruction that commercial programmers received. He summarized the situation using an allusion from the discipline of anthropology: "the lore of good coding is often passed down slowly in the ritualistic tribal dances of systems architects, analysts, project leads, and more-experienced programmers."[25] McConnell wanted to help with this technology transfer.

Code Complete focused on the topic of implementation or *construction* (coding and debugging activities), which were among the more neglected topics in computer science in the 1980s. A lot more time was being spent, he argued, on conversations about system specifications, architectural design, testing procedures, and the various tools or platforms that were in use. Construction implied a focus on designing and writing code modules, organizing control structures, finding and fixing errors, reviewing the code of others, formatting and commenting existing code, and tuning code to make it faster and smaller. Appealing to contemporary research, McConnell argued that focusing on construction-specific tasks could improve an individual programmer's productivity by a factor of 10 or 20.[26] This advice was largely independent of computer languages. Instead of languages or platforms, he focused on the characteristics of high-quality routines, code layout and style considerations, the importance of self-documenting code, quality assurance reviews, unit testing procedures, debugging strategies, and improving the performance of code.

McConnell also drew from a range of classic texts from the domains of Computer Science and Engineering. He suspected that commercial programmers may have read a language primer like Kernighan & Ritchie, or a systems primer like

24. McConnell, *Code Complete*, xi.

25. McConnell, *Code Complete*, xii.

26. McConnell, *Code Complete*, 5.

Petzold's *Programming Windows*, but few supporting works from the professional programmer's back list. These titles included:

- Gerald Weinberg's *The Psychology of Computer Programming* (1971)
- Donald Knuth's multivolume *The Art of Computer Programming* (1968 and later)
- Ed Yourdon and Larry Constantine's *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design* (1979)
- Glenford Myer's *The Art of Software Testing* (1979)
- Tom DeMarco's *Structured Analysis and Systems Specification: Tools and Techniques* (1979)
- Barry Boehm's *Software Engineering Economics* (1981)
- Jon Bentley's *Programming Pearls* (1986)
- Tom Gilb's *Principles of Software Engineering Management* (1988)
- Robert Sedgewick's *Algorithms* (1988)
- DeGrace and Stahl's *Wicked Problems, Righteous Solutions* (1990) on the software-development cycle

McConnell described these books and many others in his introductory sections, and then applied the material in 33 well-organized chapters.

In an effort to discuss the challenges of team development projects, McConnell made references to Fred Brooks' *The Mythical Man-Month*, which originally raised the alarm about the inefficiencies that often accompanied large software development projects. Brooks encountered a raft of these as he managed the design and implementation of the IBM OS/360 system in the 1960s. Fred Brooks' popular book was first published in 1975, with revised editions in 1982 and 1995. Brooks discovered that, paradoxically, the more programmers that were assigned to a project when it was falling behind schedule, the longer the project took to complete. This was because the new people added to the team, no matter how skilled, needed time to familiarize themselves with the project before they could become productive, and this learning curve required the assistance of developers who needed to stop their work to instruct the new recruits.[27] In addition, adding new team members added *complexity* to the project, because more people needed to communicate with each other, and each developer had their unique work habits and beliefs about programming. Managing software development teams was not, therefore, a process that worked according to the established rules of multiplying labor, a principle that

27. McConnell, *Code Complete*, 542–543.

was carefully worked out during the divide-and-conquer contexts of the industrial revolution. Rather, the software development process needed to be broken into distinct phases (the software development life cycle). No work on constructing the system should be attempted until all of the system architecture work was complete. McConnell went on to clarify his ideas about improving team work and product development schedules in his book *Rapid Development* (1996).[28]

Was there a way to use new or improved software tools to improve programmer productivity? According to academic research, this could be one way to improve or speed up software engineering outcomes. In *Code Complete*, McConnell investigated a few computer-aided software engineering (CASE) tools, which could help during the design phase of a project by graphically modeling the functionality of software. At the source-code level, IDEs were also bringing menu-driven, visual tools to software developers that integrated help systems, debugging, program editing, and multi-language features. A raft of commercial software tools allowed developers to browse source code quickly, search-and-replace across multiple files, assess the overall quality of code modules, track software versions, and manage deployment. However, as McConnell pointed out in 1993 and 1996, there were few comprehensive software development systems that provided *all* of these tools to the professional developer, and there were no "silver bullets" which might comprehensively speed up software development projects.[29] Adopting new tools might also bring productivity losses, because developers needed to learn the new systems and then work out any kinks.[30] Undeterred, software publishers like Microsoft, Borland, and Apple continued to build comprehensive development systems that attempted to bring the advantages of IDEs and other emerging tools to software teams. These products proliferated dramatically in the late 1990s.

Steve McConnell's *Code Complete* represented a valuable resource for self-taught programmers. Soon there were other books with similar themes that hoped to transmit successful engineering practices to wider audiences.[31] On the product development side, these efforts should be thought of as further attempts to commercialize and professionalize the learn-to-program movement.

28. Steve McConnell, *Rapid Development: Taming Wild Software Schedules* (Redmond, WA: Microsoft Press, 1996).

29. McConnell, *Code Complete*, 507.

30. McConnell, *Rapid Development*, 352, 365.

31. See Steve Maguire, *Writing Solid Code: Microsoft's Techniques for Developing Bug-Free C Programs* (Redmond, WA: Microsoft Press, 1993); and Steve Maguire, *Debugging the Development Process: Practical Strategies for Staying Focused, Hitting Ship Dates, and Building Solid Teams* (Redmond, WA: Microsoft Press, 1994).

11.6 Professional and Enterprise Development Systems

As the Windows and Windows NT operating systems gained momentum, tool makers began bundling and integrating their development systems for the PC platform, connecting disparate products to provide comprehensive solutions for their customers. An advertisement from Microsoft in *Dr. Dobb's Journal* (April 1991) provides evidence of the new trend. Readers were presented with a close-up photograph of the Microsoft C Professional Development System software box with the Microsoft Windows Software Development Kit (SDK) nearby. The caption reads, "Normally one plus one equals two. But when you add Microsoft C and the Windows Software Development kit version 3.0, things start to multiply."[32] On a facing page, the advertisement presents several colorful software boxes from the makers of Corel Draw, Aldus PageMaker, and Microsoft PowerPoint. The thrust of the ad is that the leading commercial programs written for the MS-DOS/Windows platform have been created with this pairing. The advertisement further celebrates the bundle with a claim about Microsoft's inside status: "There's a good reason for this success. Because as the creators of the Windows environment, Microsoft understands the system and its potential for personal computing."[33] Confronted with the challenging task of building commercial applications, the reader is encouraged to ask: "Why wouldn't a software developer want to use the Microsoft tools for their products?"

The answer is that the Windows software development process wasn't all that seamless in 1991, and there were many situations when Microsoft's development tools didn't work well together. Moreover, although some individual components worked fine on their own, there were entire *stages* of the software development life cycle that were left out or ignored by PC software companies. In reality, most commercial developers were mixing and matching software tools from different publishers to do their work, sometimes writing their own tools to address the shortcomings of individual products. The problem was exacerbated when *multiplatform* development became an aspiration for many PC software firms. As we have already discussed, businesses routinely supported multiple operating systems in their organizations, but the interoperability challenge grew complex when Microsoft Windows, OS/2 Presentation Manager, Windows NT, Unix, and Mac OS all co-existed in the marketplace. The move to Internet-based platforms exacerbated this problem in the mid-1990s, and by the 2000s, smart phones and tablets also arrived on the scene, requiring custom versions of the major operating systems for these products.

32. Microsoft advertisement (2-page spread) in *Dr. Dobb's Journal* 16, no. 4 (April 1991), 1.

33. Microsoft advertisement, *Dr. Dobb's Journal*, 1.

One solution to the challenge was the development of comprehensive programming *suites*, which offered numerous languages, tools, and libraries integrated into one common IDE. With these complex systems, professional software developers could potentially manage many of the tasks associated with the software development life cycle for different platforms, and they could share information and source code among team members. These systems were carefully designed and implemented as integrated software products, often in close association with the release of new operating systems.

The first Microsoft versions of commercial, integrated development suites came in 1997, with the release of Microsoft Visual Studio 97 Professional Edition and Microsoft Visual Studio 97 Enterprise Edition. These products sold for $999 and $1,499, respectively. If customers owned earlier versions of Microsoft's development tools, they could purchase upgrades at a substantial discount. Microsoft announced the products via dozens of "Developer Days" events, beginning in March 1997. The company made an effort to make the announcements appear global in scope, and they followed a "global ready" development strategy to release localized versions of Visual Studio 97 soon after the English-language release. Impressively, Visual Studio events were held in 45 countries, including Argentina, Australia, Canada, Egypt, France, Germany, India, Israel, Thailand, and the U.S. This international emphasis was a stark contrast to earlier U.S.-focused releases of the company's compilers and development tools.

Some 45,000 software developers received hands-on exposure to Visual Studio at the 1997 Developer Days events. As Chapters 9 and 10 have emphasized, Microsoft's programming technologies were not always easy to learn, and the events allowed aspiring developers an opportunity to network and compare notes about the new systems. The training events were more in-depth than COMDEX, the West Coast Computer Faire, or similar trade shows. But the company realized that every developer brought to Windows application development would help to solidify the platform against the inroads of competing rivals. Eventually, the company broadened Developer Days into a series of software development conferences, which Microsoft used to advertise and support its products for each release. These events included the Professional Developers Conference (PDC), TechEd, MIX, and Build. Through these commercial venues, the learn-to-program movement continued in its corporate manifestation.

What was the platform strategy behind Microsoft's Visual Studio release? Integrating components into a flagship corporate product was one important goal, and the company followed a "suite" strategy that it had recently implemented with Microsoft Office, the integrated application suite for Windows and the Macintosh. In 1995, Bill Gates also circulated the infamous "Internet Tidal Wave" memo at

Microsoft, directing the company to pivot rapidly to Internet-based technologies and products. Visual Studio 97 offered the first fruits of this web-based approach to software development on the PC platform. The suite included support for deploying client/server applications on the Web, as well as the traditional desktop platforms of MS-DOS and Windows. Gates articulated support for a multiplatform approach in a press release at the time of the Visual Studio 97 announcement:

> We have always been committed to providing developers with the best tools and training to take advantage of the rapidly changing computing infrastructure. Our vision for applications development is to make it easy for developers to integrate client/server and the Internet, as well as make it easy to build robust, multitier enterprise solutions that can take advantage of existing systems. Developers are the heart of the industry and very important customers for us.[34]

Clearly, adapting to a multitier, multi-platform world was important for Microsoft, but the complexity brought many challenges for software developers. The techniques used to develop applications for MS-DOS, Windows, and Internet-based web browsers were very different. However, the company would stick with this strategy for the next decade, and in some ways it characterizes Microsoft's decision making into the 2010s.

What components were in the Microsoft development suite? The base product, Visual Studio 97 Professional Edition, included the following programs:

- Version 5.0 of the Microsoft Visual Basic programming system
- Version 5.0 of Microsoft Visual C++, the development system used to build commercial Windows applications in C and C++
- Version 1.0 of Microsoft Visual InterDev, a development system for building and managing data-driven web applications
- Version 1.1 of the Microsoft Visual J++ programming system, for creating cross-platform Java Applets and applications and integrating Java with ActiveX technologies
- Version 5.0 of Microsoft Visual FoxPro, a relational database management system with an object-oriented programming language
- Microsoft Developer Network Library (MSDN), an online source of official product documentation and developer training

34. Bill Gates quoted in "More than 45,000 developers see public unveiling of new Microsoft Visual Studio 97 at Developer Days events worldwide," Microsoft Corporation, March 18, 1997.

A selection of language compilers was included because Microsoft believed that many programmers were using more than one language to create their application solutions. Users could switch from one programming language to the next within the same Visual Studio IDE.

Customers who purchased Visual Studio 97 Enterprise Edition received the Standard Edition products plus tools that would help them with team-based projects and enterprise-wide computing. These programs included:

- Microsoft Transaction Server 1.0, which supported larger distributed applications
- Microsoft SQL Server 6.5, a relational database management system
- Visual SourceSafe 5.0, a version control system for commercial contexts
- Microsoft Visual Modeler, a modeling tool for representing large-scale component-based applications

Enterprise-wide projects are software applications that are widely deployed and used throughout an organization. Microsoft intended these applications to be run on the Windows NT Server operating system, but they needed to function *within* the existing computing infrastructure. In short, enterprise applications needed to be highly "scalable," or easily expanded and/or upgraded based on the demands of the company's changing needs and their global network. In this way, the commercial contexts of IT programming in the late 1990s outstripped the demands of the rudimentary compilers and SDKs sold by Microsoft and other publishers a decade earlier. No one person on an enterprise development team would use all of the products in the Visual Studio development suite, but the business proposition was that the entire company should adopt these tools, so that incompatibilities and training problems would not ripple through the organization. Over time, Visual Studio Enterprise Edition also attempted to address the needs of those who were engaged with infrastructure planning, data access, application design, project management, implementation, solution testing, deployment, security, and much more.[35]

How did Visual Studio fair in the commercial marketplace? Within a year, Microsoft announced that Visual Studio Enterprise Edition had been adopted by over 90% of Fortune 1000 companies in America as the leading development

35. For more on contemporary best practices for designing and implementing enterprise software systems, see *Enterprise Systems Integration*, ed. John Wyzalek (Boca Raton, FL: Auerbach, 2000). This developing world was the market that Microsoft was trying to target with its enterprise development products.

suite for business applications and multi-tiered "real-world scenarios."[36] Microsoft claimed that their enterprise-wide approach was supporting companies that were building solutions across a wide range of operating systems, including Windows 3.1, Windows 95, Windows NT, Apple Macintosh, and Unix. They also pledged to help corporate developers solve enterprise and corporate computing issues in the future, including "analysis, design and modeling, testing and quality assurance, configuration and process management, and legacy system connectivity."[37] Within about 18 months, they released Microsoft Visual Studio 6.0 Enterprise Edition (August 1998), a comprehensive development suite that integrated even more features for corporate developers. Gradually, it helped Microsoft gain additional ground in the compiler and operating system wars.

11.7 Commercialization

As Internet-based client/server computing arrived, larger software publishers like Microsoft shifted their marketing emphasis from individual programmers, hobbyists, and students to America's Fortune 1000 corporations, which employed legions of software developers to handle their corporate IT needs. Microsoft continued to sell software to individual programmers and consumers, but they also recognized that a major transition that was taking place in the global software business. Powerful new hardware and massively scalable operating systems like Windows NT Server allowed corporations to replace their mainframe and minicomputer systems with banks of network servers running Windows NT, Unix, and other PC-based operating systems. This cost-shifting measure further fueled the commercialization of PC programming in the late 1990s. In many ways, the PC software industry now faced the daunting distributed computing scenarios that that had challenged and inspired computer architects in the 1960s and 1970s. Companies like Microsoft, IBM, Sun, and Oracle chose to emphasize the needs of their corporate customers over those who were learning to code on new systems.

How did the forces of commercialization impact the learn-to-program movement? In the 1970s, the "PC Revolution" brought new computing platforms to millions of Americans. Out of necessity, many wrote computer programs on the new devices because there was a dearth of application software available. Thanks to the work of programming evangelists like Bob Albrecht, Mitchell Waite, Dian Crayne, and Ray Duncan, thousands of students and hobbyists learned to code and build interesting applications for the CP/M, MS-DOS, and Macintosh platforms. Although

36. Microsoft Corporation, "Corporations Embrace Visual Studio 97 Enterprise Edition for Mission-Critical Application Development Projects," February 3, 1998.

37. Microsoft, "Corporations Embrace Visual Studio 97."

the advocates for K-12 education in the U.S. lost faith in the transformative power of programming in schools, a new raft of programming tools restored confidence in the movement to the masses. New machines based on the 80386 and 80486 microprocessors made it clear that individuals who wrote their own programs could tap into powerful engines that offered the prospect of future employment or software entrepreneurship.

The transformative power of personal computing influenced America's corporate computing cultures at a slower pace, however. It was not until the late 1990s that the PC software industry became a substantial commercial force in the American economy, finally surpassing the corporate/mainframe software services industry. The impact of commercialization was obvious when the rising hardware and software platforms became stable enough to support major corporate investments in business applications, development tools, and IT/MIS infrastructures. Only at this point could the largest business organizations justify the costs (and attendant risks) of fully investing in PC-based systems. One of the crucial products that facilitated this transition was Microsoft Windows NT Server, a scalable client/server operating system that made it possible to replace existing mainframe and minicomputer infrastructures with distributed systems in America's leading corporations.[38]

By 2003, the shrink-wrapped or "packaged" software industry in the U.S. produced revenues of $178 billion. At that time, there were more than 10,000 businesses in America building specialized commercial applications.[39] To complete this work, there were approximately 1.6 million professional programmers, computer scientists, and systems analysists laboring in the U.S. workforce. A majority were self-taught programmers, who learned their computing skills outside of the traditional, 4-year college system. But the drum beat of economic opportunity continued. A new wave of software "evangelists" advertised the merits of the emerging hardware/software platforms with the fervor of missionaries, spreading the "good news" and attracting converts and disciples for the new systems. Commercial trade shows like COMDEX and Macworld Expo offered tantalizing glimpses of the newest technologies, and the industry luminaries who sold the products used their charisma and influence to build their platforms. Those who chose to learn programming often did so in the context of professional development tools and corporate "messaging" events like Microsoft Developer Days, Teched, and Build. These trade shows indoctrinated new developers into the commercial worlds of programming, accompanied by logoed tee-shirts, water bottles, and platform messaging.

38. For more on the transition to this infrastructure, see Helen Custer, *Inside Windows NT* (Redmond, WA: Microsoft Press, 1992), 3.

39. Evans, Hagiu, and Schmalensee, *Invisible Engines*, 84.

Code Nation has often stressed the importance of programming primers and "how-to" books for the transmission of software development practices. But were these books impacted by commercialization as much as the software development tools? The short answer is "yes," with the most obvious impact being the proliferation of topics and the close alignment of programming skills to specific products and the demanding product schedules of the software industry. In an earlier era, bestselling primers like Kernighan and Ritchie's *The C Programming Language* taught the fundamentals of C programming by focusing on a relatively small number of features and concepts. In just 228 pages, the authors taught core principles that they reinforced through examples that were independent of the underlying hardware.

In the new realms of enterprise computing, however, aspiring software developers learned the ropes by dividing and conquering. First, a programmer would learn the fundamentals of a given computer language such as Java, C++, or Visual Basic. This would almost always be done in the context of a specific compiler or development system. Next, students would learn platform-related skills, such as how to create an appropriate user interface for the Apple Macintosh, or how to use the Win32 API. In a RAD system like Visual Basic, aspiring developers had to master both visual design concepts and the underlying features of the Visual Basic programming language. Visual and syntactical elements were also a requirement for C/C++ programmers who were working with the Windows SDK. For example, developers needed to populate and control application windows, menus, dialog boxes, and other user interface elements. Then they needed to manage memory, inter-process communication, multitasking, peripherals, and the attributes of an advanced operating system. On top of this, corporate and enterprise developers required additional skills and tools. They needed to learn about team-based work flows, how to maintain a code base, how to plan for localization, testing best practices, how to document the software, and the steps necessary for distribution and deployment.

Computer book authors, trainers, and publishers attempted to equip developers for all of these tasks by preparing a range of vertical market titles for most of the major software platforms. One creative publisher, Wrox Press (founded in 1992), created an effective line of books to address the commercial manifestations of the learn-to-program movement. Their "Programmer to Programmer" series featured carefully scaffolded or "tiered" titles with content that began with introductory topics and then moved deeper into the product as developers gained more experience. (See Figure 11.7.) Rather than teaching computer science fundamentals, as earlier books did, Wrox titles emphasized the commercial features of a new operating system, product, or language. The publisher also offered "Public Beta" books that provided short "walkthroughs" of products still in development, publishing

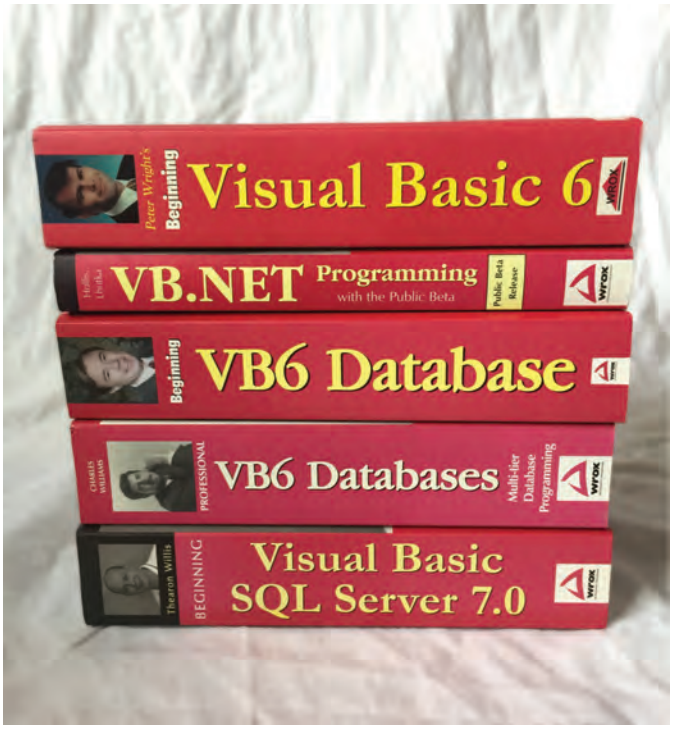


Figure 11.7 **Examples of the "Programmer to Programmer" book series published by Wrox Press in the late 1990s. Wrox programming primers followed a "scaffolded" approach, which gradually introduced the components of a commercial technology such as Visual Basic 6, released in 1998.** (Photo by Michael Halvorson. Images courtesy of Springer Nature AG)

technical descriptions of compilers that had not yet been released, so that consumers could get a jump on their rivals (and help software publishers debug their software). Each of the books was written by a professional software developer with industry experience, and they emphasized skill-building for IT careers as well as traditional software development jobs. Importantly, the titles also flagged the market segment that they were designed for. Marketing copy on the back of *Beginning Visual Basic 6 Database Programming* (1998), by John Connell, explains the tiered learning approach in this way:

> Wrox Beginning Guides are expertly crafted to make learning fundamental programming techniques easier than you think. Whether you're taking your first steps in programming or broadening your skills with new techniques, Wrox books guarantee a carefully structured tutorial format that will guide you through all the techniques involved... These projects take you right up

> to the point where you can develop professional applications to be proud of. Our aim is to make you successful by sharing the knowledge of experienced programmers with you at every stage in your career.[40]

Microsoft Corporation scaffolded its popular training and certification programs in the same way in the 1990s, hoping to provide a clear path through what otherwise looked like a complex world of unfamiliar proprietary technologies. The company introduced professional certification in 1992, with the modest goal of assisting computer professionals to demonstrate their new IT skills to potential employers. The first certification exams were published for Microsoft Windows 3.1, Microsoft LAN Manager, and SQL Server. Although industry certification was not designed to replace academic credentials or the infamous "white board interviews" that confronted developers in job interviews, it became a popular mechanism for gaining credentials.

The first certification levels included Microsoft Certified Professional (MCP), Microsoft Certified Systems Engineer (MCSE), and Microsoft Certified Solutions Developer (MCSD). The candidates for these achievements prepared by purchasing Microsoft software and training manuals, and then they took exams prepared by industry experts. Like the "Programmer to Programmer" series published by Wrox, Microsoft's certification materials emphasized how an IT professional might integrate commercial products to build enterprise-wide solutions, rather than encouraging problem-solving or computational logic skills. The system of scaffolded certification courses and training materials became very popular, and it continued into the Internet age through an assortment of on-line and instructor led courses.

The commercialization of the learn-to-program movement was perhaps an inevitable outcome of the rapid expansion and economic flourishing of the PC industry. Commercialization gradually shifted the emphasis of software companies from the hobbyists and solo entrepreneurs who tinkered with early PC systems to the corporate developers who managed enterprise-wide systems. This transition took place most dramatically in the 1990s, as the PC industry expanded and became an undeniable force in the U.S. economy. In this context, the leaders of the learn-to-program movement devoted their energies to teaching the corporate manifestations of popular programming products, including Borland Turbo Pascal, the Windows Software Development Kit, and Microsoft Visual Studio. Each product was carefully positioned in the marketplace and offered as a "solution" that would help aspiring commercial developers to build successful applications.

---

40. John Connell, *Beginning Visual Basic 6 Database Programming* (Birmingham, UK: Wrox Press, 1998), back cover copy.

During this transition, the rudimentary interpreters and compilers of the 1970s and 1980s ("Tiny BASIC" and "Tiny C") were replaced by development suites with enticing features, menu-driven IDEs, tools for the major platforms, and reusable code for enterprise developers. To respond to this complexity, authors and publishers developed a scaffolded approach to introducing programming skills, publishing numerous books and articles to help aspiring programmers learn the skills they needed for commercial development. The era's engineers and decision makers also had access to commercial computing trade shows, which mixed members of the business and technical communities with journalists, industry luminaries, and members of the general public. Learning to program went hand in hand with the vitalities and agendas of American consumerism.

In the Afterword, I'll conclude *Code Nation* with a look at how the Internet has impacted computer programming and technical publishing in the U.S. I'll summarize how new computer programmers learned to program on the World Wide Web, and contrast web-based learning strategies with print-based methods. The Afterword also serves as a summary and conclusion for *Code Nation*. I'll review the trajectory of the learn-to-program movement and its significance in American culture, past and present, including a look at modern expressions of the movement's ideals.

Afterword: Programming in the Internet Age

"Computers can be the technical foundation of a new and dramatically enhanced literacy, which will act in many ways like current literacy and which will have penetration and depth of influence comparable to what we have already experienced in coming to achieve a mass, text-based society."

Andrea A. diSessa, *Changing Minds: Computers, Learning, and Literacy* (2001)

"It doesn't matter to what degree an individual learns to code; that knowledge does not need to lead to professional programming. The goal is for the general population to pierce the computing veil to demystify algorithms; to know that code has biases, that programs are written by human beings and can be changed by human beings..."

Ellen Ullman, *Life in Code* (2017)

Code Nation has examined the rise of computer programming and the social, technical, and commercial worldviews that coalesced to form a new type of computing culture in America. A central part of this story is the learn-to-program movement, which germinated in government and university labs during the 1950s, gained momentum through counterculture experiments in the early 1970s, became a broad-based literacy movement in the late 1970s and 1980s, and was transformed by commercialization in the 1990s and 2000s. The learn-to-program movement sought to make computers more understandable, imprint useful technical skills, establish shared values, and offer economic opportunities for computing enthusiasts. The movement also supported user communities, schools, and emerging industries, many of which benefited greatly from the utility provided by digital electronic computers.

The scope of the learn-to-program movement can be measured in a variety of ways. One outcome was a dramatic increase in the number of professional coders who could design, create, and maintain software. In 1957, there were approximately

15,000 programmers employed in the U.S., a figure that accounts for approximately 80% of the world's developers. By 2003, there were approximately 1.6 million professional programmers, computer scientists, and systems analysists registered in the U.S. workforce. Since that time, software creation has become a global phenomenon, with millions of people learning to write computer programs in a variety of contexts. For example, in 2014, there were approximately 18.5 million software developers in the world, of which 11 million can be classified as professional programmers and 7.5 million as hobbyists. Thanks to schools, coding boot camps, and robust non-profit organizations, these numbers increase daily. So, too, are warnings that if a person does not heed the call to "learn coding now," they will miss something of what the global digital economy has to offer.

Despite a shared exposure to computational thinking, programming has also changed as an intellectual activity. In the 1950s and 1960s, the first programmers were involved with building and maintaining military systems, designing algorithms for scientific research, tracking census data, and implementing data-processing schemes for agencies and corporations. Today's software developers are involved with an even wider range of activities, including consumer software, scientific research, video game programming, artificial intelligence, information publishing, digital communication, data mining, education, art, music, streaming services, gambling, medicine, sports, and myriad undertakings that require the intensive use of computers. Many programmers create or maintain software as part of their regular employment, while others write code for volunteer organizations, recreation, school work, or as an aspect of their personal development.

Programming personal computers (PCs) was not a radically new activity. As many scholars have noted, before microcomputers and PCs there were powerful mainframe and minicomputer systems that offered software developers a rich digital experience coding in assembly language, FORTRAN, BASIC, C, and other languages. However, the development of PCs did allow for the rapid deployment of new commercial platforms, which quickly grew in power and sophistication in the 1980s and 1990s. The commercial "PC Revolution" began with the Tandy TRS-80 (1977), the Apple II (1977), the IBM PC (1981), and the IBM PC XT (1983). Soon more advanced systems appeared in the marketplace such as the IBM PC AT (1984), the Apple Macintosh (1984), and powerful "clone" systems that featured Intel 80386 and Intel 80486 microprocessors. These devices contributed to the proliferation of PC-based communities that shared source code, bought software and peripherals, read computer books, and met together in user groups and at trade shows. The result was a collection of commercially viable platforms built around hardware brands, software, operating systems, product marketing,

and dedicated user communities. Among the platforms that emerged, this book focuses on the MS-DOS, Microsoft Windows, and Macintosh platforms. I have also indicated points of intersection with other important user groups and technologies, including CP/M, OS/2, Unix/Xenix, and early time-sharing systems.

Computing Mythologies

Code Nation draws attention to four foundation mythologies that took shape in the early computer industry and subsequently influenced the learn-to-program movement. Computing mythologies are socially-constructed memories that can carry important historical and cultural information. They also act as social markers, transmitting ideas, beliefs, and worldviews to the members of a movement and future generations. First, the myth of ongoing "crisis" in the computer industry related to the complexity of software systems has led to regular calls for the introduction of software engineering techniques and tighter control over programming practices. This crisis-mentality first appeared in the 1960s, and it was revisited during the shift to graphical user interface (GUI) programming and enterprise computing in the second and third phases of PC development. A second myth is a popular narrative about rejecting corporate and military tendencies in computing and embracing a democratizing, countercultural ethos regarding convivial tools and technology. This myth took shape alongside anti-war protests in the 1960s, and it continued with calls to put computers in the hands of regular people. Stewart Brand, Ted Nelson, Robert Albrecht, Judith Milhon, Dan Gookin, and my own work sits in this tradition, and it is most prominently visible today in Code.org's Hour of Code movement.

A third computing mythology relates to the scholarly objectives of the discipline of computer science, which took shape in the 1950s and 1960s in America. Over the next decade or two, many computer professionals came to believe that academic computer scientists were occupied primarily with theoretical problems related to computational logic, algorithms, and engineering principles, rather than the practical skills needed in the computer industry. For this and other reasons, many computer professionals took jobs in software development without studying computer science or computer engineering in college, and often missed the gains of scholarly research produced by academics. Likewise, the programmers who *have* benefited from college degree programs and theoretical research sometimes disparage those who were not so fortunate, deepening fissures that separate academia, industry, and various self-taught communities. Grace Murray Hopper spoke about this in 1978 when she said, "By and large, the people concerned with the large computers have totally ignored the so-called hobbyist community, though I would

point out they are not a hobbyist community: they are small businessmen, doctors, lawyers, small towns, counties! They are a very worthwhile audience."[1]

Fourth, there are several mythologies related to what is often called "the PC Revolution," a phrase that captures much of the excitement surrounding the creation of the first microcomputers and PCs in the 1970s and 1980s. This term draws attention to vital energies in American capitalism and the computer industry, but it also lionizes the experience of PC users and entrepreneurs over professionals working in other areas of digital computing. I have tried to minimize this rhetoric and the role of "founding biographies" in this book, while also appreciating the important ways that personal computing has contributed to the formation of American programming culture and the software industry.

To these sustaining computer myths, which shaped and influenced the learn-to-program movement, I would like to add a fifth that came into view by the 1990s—a belief that personal computing has become one of the premier engines of capitalism in the U.S. economy, a vital source of innovation and creativity that is linked to what some call the "American Project" and popular understandings of the American Dream. The economic piece of this myth can be appreciated through a few financial milestones. By the late-1990s, the PC software industry finally surpassed the corporate/mainframe software services industry in the U.S. in terms of revenue. This achievement provided important evidence that PC software was lucrative and soon to become a global phenomenon. By 2003, the shrink-wrapped software industry produced revenues of $178 billion, and there were more than 10,000 businesses in America creating specialized software applications. Commercial trade shows like COMDEX and Macworld Expo offered tantalizing glimpses of these new products, and software evangelists advertised the emerging platforms with the fervor of missionaries, spreading the "good news" and attracting new "converts." Americans poured their hopes and dreams into software and its promise of a better world through productivity and connectivity.

Each of these computing mythologies is a partial truth, of course, a form of national dreaming about technology and capitalism that glorifies the software creation process and which has been used to push forward the nation's shared technical identity. Structurally, computerization movements require ideologies of this type to sustain the movement when the road ahead becomes difficult or unclear. Computer pioneers like Grace Hopper, Ted Nelson, Alan Perlis, Seymour Papert, and Bill Gates regularly provided glimpses of these mythologies to keep the

1. Grace Murray Hopper, "Keynote Address," ACM History of Programming Languages Conference, June 1, 1978, in *History of Programming Languages*, ed. Richard L. Wexelblat (New York: Academic Press, 1981), 22.

learn-to-program movement on track.[2] Through writing and creative activity, they sketched a new world order that might come into view when the rhythms of computational thought were more deeply infused in American life. Motivated by these ideas, scores of talented programmers, teachers, authors, and entrepreneurs introduced programming concepts to the masses, sharing with them the utopian ideal about personal computing.

The Commercial Internet

Of course, sometimes computer visionaries made mistakes. Somewhat famously, Bill Gates and the majority of his peers in the PC industry underestimated the significance of the commercial Internet when it arrived in the early 1990s. How did this happen? The following section summarizes the technical underpinnings of the commercial Internet and its impact on programming practices and the learn-to-program movement.

Connecting PCs in a significant way over phone lines began with bulletin board systems (BBSs) in the early 1980s, which utilized dial-up message boards, server software, and commercially available modems. These interesting systems attracted advanced users and sometimes hackers (see Chapter 7), but BBS culture was not seen as an extraordinary phenomenon in the wider world of computers. The Advanced Research Projects Agency Network (ARPANET) had also been available in limited ways for research and government use since the 1960s, and many corporations were experimenting with distributed computer networks as a way to exchange information through shared digital pathways.[3] By the late 1980s, most U.S. software companies were content to network their computers *internally* (within an organization), while providing limited access to shared (public) networking sites in the outside world. One exception was that consumer software companies routinely provided phone and fax machine support for their products, as well as information through emerging dial-up networks such as CompuServe and America Online (AOL). These network providers offered paying customers many of the features that would later become popular on the World Wide Web, including email, file transfer capability, news feeds, shopping opportunities, and discussion forums. Technology

2. Bill Gates' book, *The Road Ahead*, draws specifically on utopian ideals in its call to prepare for an exciting future enhanced by computer programs and Internet-related technologies. See Bill Gates, *The Road Ahead* (New York: Viking/Penguin, 1995; rev. ed., 1996). A similar book, highlighting the importance of e-commerce and corporate decision making, is Bill Gates, *Business @ the Speed of Thought: Using a Digital Nervous System* (New York: Viking/Penguin, 1999).

3. For an excellent summary of early distributed networking technologies and their use, see Andrew S. Tanenbaum, *Computer Networks* (Englewood Cliffs, NJ: Prentice-Hall, 1981).

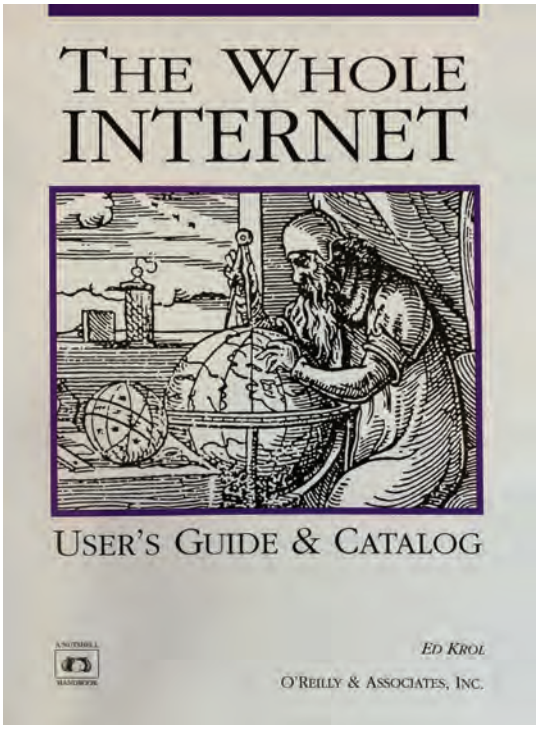


Figure A-1 **Ed Krol's *The Whole Internet User's Guide and Catalog* (1992) was one of the first trade books to sample the networking tools and utilities available on the Internet. It was published just before the commercial "Internet Revolution" took place.** (Image courtesy of O'Reilly Media, Inc)

companies also hoped to use these services as a way to offload some of the product support calls that they were receiving as customer bases grew in the late 1980s and early 1990s. However, the services were not available in all locations and they typically involved long distance calls and monthly fees.

Fascinatingly, the emerging Internet was not seen as a revolutionary technology for the masses, even months before its commercial introduction in the U.S. A pioneering computer book that explored this nascent online world was Ed Krol's *The Whole Internet User's Guide and Catalog* (1992).[4] (See Figure A-1.) Krol's reference was situated in the tradition of *The Whole Earth Catalog*, but the computer book did not adopt the oversized layout of Stewart Brand's counterculture compendium (see Figure 2.5). In fact, this technical book was mostly pitched for power users and advanced hobbyists, who O'Reilly believed might appreciate a survey of dial-up tools including Telnet, Ftp, Gopher, and USENET. An important observation that

4. Ed Krol, *The Whole Internet User's Guide and Catalog* (Sebastopol, CA: O'Reilly & Associates, 1992).

the book made was that there was not one networking system to learn on the Internet, but several. Most of these programs were command-line oriented and cryptic in their presentation. A telling section title in the book, "What if I don't know UNIX?" seemed to address common fears about getting started with this online world and its illusive protocols.[5]

As many know, the consumer "Internet Revolution" gradually arrived over the next year or two. The Mosaic web browser was released in 1993, followed by Netscape Navigator in 1994. These software applications put a GUI on the cryptic format of the Internet and popularized its use via the World Wide Web. These programs allowed customers to move through an exciting world of images and digital texts using a mouse-based, point-and-click interface. Although Netscape Navigator did not gain a significant market share until 1996, Bill Gates and his Microsoft colleagues had seen enough. In May 1995, Gates circulated the famous "Internet Tidal Wave" memo at Microsoft, directing the company to pivot rapidly to Internet-based technologies and products. Microsoft released Internet Explorer and Microsoft Network later that year, and by the mid-1990s there were numerous Internet-based browsers and networking systems in the PC marketplace. From an economic point of view, the mid-1990s initiated a new phase in the computer industry in which Internet-based products and services struggled for dominance with traditional goods and services, often disrupting traditional industries.

For computer programmers, the commercial Internet brought many changes because application development for web browsers was significantly different than creating programs for MS-DOS, Windows, or Mac OS. To make an application *minimally* web-aware, a first step might be to give users the ability to download files from a web server to a client computer. A *server* is a computer running on the Internet that maintains or hosts a web page and other Internet services. A *client* is a computer with access to the Internet that requests information from a server using various protocols. Client requests for information might originate from a web browser program or service, or from an application written in C, Visual Basic, Java, or another language. A Visual Basic programming book that tried to present this evolving world to intermediate-level programmers was Carl Franklin's *Visual Basic 4.0 Internet Programming*, published by John Wiley & Sons in 1996.[6] This type of Web programming book was very hard to write and revise because the underlying technologies changed so dramatically from one version the next. (For example, an author needed to track changes to the web browser software, the Visual Basic compiler and IDE, and the host operating system.)

5. Krol, *The Whole Internet User's Guide*, 5.

6. Carl Franklin, *Visual Basic 4.0 Internet Programming* (New York: John Wiley & Sons, 1996).

A more sophisticated Internet application could be written using hypertext markup language (HTML), a system of formatting codes for presenting information that virtually all web browsers could process and display. Internet applications resided in a variety of locations. For example, an application might reside on a web server and be launched from a web browser. Or, an application might run in a local web browser, using resources entirely from a user's computer, but occasionally gathering information from the Internet as it completed its work. Either way, Internet-aware applications needed to be adept at communicating back and forth with Internet-based computers; they needed to be proficient at managing data (so that information was not lost during transactions); and they needed to function on a variety of web browsers. The makers of PC programming systems needed to update their development tools accordingly, providing these features to commercial and hobbyist developers who wanted to expand their work to Internet-based contexts.

Sun Microsystems addressed the challenge of client/server development by releasing the Java programming language in 1995, which addressed some of the opportunities of the new Internet platform. Like C++ and Smalltalk, Java was class-based and object oriented, and it could be used as a general-purpose programming language to create software for a variety of tasks. However, the language was also designed to minimize implementation dependencies, so that developers could "write once, run anywhere." If Java programs were constructed carefully, software developers could move their modules from one platform to the next without rewriting or recompiling the code. Although Java had many possible applications, the language became especially popular for designing server-side (back end) Internet applications.

On the client side (or front end), the JavaScript language from Netscape Communications became a popular programming tool for creating interactive web pages. JavaScript was also released in 1995, and it had some similarities to Java, including its support for the event-driven, object-oriented programming paradigm. However, JavaScript programs were designed to be run *inside* a web browser that supported JavaScript features. Looked at comprehensively, client-side development required three essential web technologies: HTML, to specify the content of web pages; cascading style sheets (CSS), to define how the web pages are presented; and JavaScript, to define the behavior of web pages. If the web developer so desired, they could use a competing scripting language in place of JavaScript.

Microsoft responded to the challenge of developing Internet-based applications by releasing Microsoft Visual Studio 97, which represented the company's first effort to provide comprehensive tools for the PC platform that included web technologies. This suite provided integrated support for deploying client/server applications on the web, as well as the "traditional" PC platforms of MS-DOS, Windows, and

Windows NT. When the company released Visual Basic 6.0 in 1998, the product also included support for dynamic hypertext markup language (DHTML), an Internet technology based on the Microsoft component object model (COM) specification and guidance from the World Wide Web Consortium. This allowed Visual Basic applications to run on the Internet and run inside web browsers, although only a subset of Visual Basic features was allowed for Internet applications. To create the web applications inside Visual Studio, programmers needed to use the DHTML Page Designer and a new set of Toolbox controls.[7]

In summary, it became obvious that a new set of complexities awaited PC programmers who hoped to create Internet-based applications in the late 1990s. The reality was that many organizations chose to support multiple platforms in their future development work, supporting web-based and PC-based applications at the same time. The programming tools did not always make it easy to port an application from one platform to the next. However, computer book authors and magazine columnists responded again to the challenge, and they produced dozens of interesting programming titles on HTML, XML, Java, JavaScript, C#, and Visual Basic. Listing them here goes beyond the scope of this book.

It seemed like the complexities of the Internet might be managed through the current teaching and learning infrastructure in the computer publishing industry. By the early 2000s, though, something had changed in the publishing business.

Disruption

The early 21st century was not kind to computer book and magazine publishers. Between 2000 and 2010, the book and magazine industries declined drastically in terms of units and revenue, and many publishers simply went out of business. The problem was not a shortage of new programmers to instruct—the software industry continued to expand and flourish, and the momentum was extended by new user products including tablets, MP3 players, and smart phones. Rather, the issue was that computer book and magazine publishing went through the same grinding transformation that most traditional media did when the commercial Internet arrived. Telephone, television, radio, film, and print media—all were impacted in similar ways when the Internet widened information flows, fractured traditional monopolies, changed cost structures, and brought forward new products.

Computer book publishing dropped dramatically in 2001, when the overall revenue for technical books published in the U.S. declined by almost 20% in a year and

7. For a walk-through of the steps involved, see Michael Halvorson, *Microsoft Visual Basic 6.0 Step by Step Professional* (Redmond, WA: Microsoft Press, 1998), 507–28.

continued to fall. Reflecting on the turbulence, Mike Hendrickson, Associate Publisher of O'Reilly Media, wrote: "The computer book market cratered in 2001 [in the U.S.], shrinking twenty percent a year for three years until it stabilized in 2004 at about half the size that it was in 2000."[8] The market recovered a little in 2005 and 2006, only to begin another decline, so that by 2008, the market was about 25% smaller than it had been in 2004.

The long-term trend was clear—computer book and magazine publishing was no longer the growth engine that it had been in the U.S. economy; rather, the markets were shrinking fast. Although there were still thousands of computer books in print, the overall market for technical books would soon be just 10%–20% of what it had been in the late 1990s. Numerous computer magazines also ceased publication, or shifted to online editions that had much smaller user bases. The crisis brought mergers, downsizing, the departure of big-name authors, and the collapse of several bookstore chains. An early victim of the restructuring was Borders Group Inc., which went out of business in 2012. Borders had been a successful book retailer in the U.S. with bookstores in shopping malls throughout the country. But the company could not compete with Amazon.com and other Internet retailers, which offered lower prices and a huge selection of online products.

How did successful book publishers adapt? The companies that did not close experimented with mergers and strategic partnerships to keep revenues up and they tried to increase their market share. (Although the publishing pie was shrinking, publishers hoped to gain bigger slices of the remaining pie by merging, preparing for the day when sales volumes might return.) For example, Pearson acquired Peachpit Press and Sams, and Wiley acquired Sybex, Wrox Press, and the For Dummies series. Microsoft Press entered into a multiyear publishing arrangement with O'Reilly Media.[9]

Book publishers also experimented with new methods, including e-book publishing and launching comprehensive websites. Although digital books had many advantages (you could download them immediately and use search tools in them), the first digital titles were relatively easy to steal or "pirate" on the Internet, and off-shore distributors also sold many titles illegally and at rock-bottom prices. Specialty formats were soon developed that were harder to copy or download illegally, including EPUB, Mobipocket, and iBooks. By 2010, the e-book revenues for some

8. Mike Hendrickson, *State of the Computer Book Market 2008* (Sebastopol, CA: O'Reilly Media, 2009). Available as an e-book sold through retail channels or the O'Reilly Media website, http://radar.oreilly.com/2009/02/state-of-the-computer-book-mar-17.html.

9. A summary of the mergers is provided by Hendrickson in *State of the Computer Book Market 2011*, Part 3: The Publishers.

of the more proactive publishers (including O'Reilly Media) were actually exceeding what printed books were bringing in. Using data from Bowker research (a ProQuest affiliate), Mike Hendrickson concluded that by 2011, some 20% of U.S. adults had downloaded at least one digital book and paid for it. As the transformation continued, customers shifted away from traditional publishers to a bevy of creative learning resources. These included blogs, wikis, videos, online databases, subscription sites, and custom software applications. New resources soon became available on the World Wide Web, including YouTube videos, Lynda.com tutorials, the Khan Academy, and numerous coding boot camps. By the early 2000s, hundreds of programmer/entrepreneurs published their own websites, blogs, and newsfeeds about software development and how a person might learn to code. For those who wanted programming instruction, a wealth of new resources was suddenly available, even as the older systems sunsetted.[10]

Successful corporations like Apple Computer, Microsoft, Oracle, Google, and Amazon also joined the party. They took the lead in building corporate websites that helped their customers learn how to develop software products. Through the Microsoft Developer Network (MSDN), for example, Microsoft provided training and support for new and experienced programmers through web pages, forums, and information hubs for individual products. Microsoft customers were able to download software, get product support, and complete a selection of online tutorials related to programming. Within the forums, it was also possible to receive "recognition points" or "achievement medals" for posting advice and offering support to fellow users. In the 2000s, several companies developed these point systems to motivate programmers to answer questions and provide a peer perspective on how to solve problems.

The employees of software companies also took a more visible role in product forums, providing developer-to-developer support that complemented or took the place of traditional primers and reference manuals. By supporting software development tools in 24/7 online environments, the most successful software publishers gradually adapted to the new world of global Internet connectivity. Baldur Bjarnason summarized the transition to Internet-based learning systems in this way:

> Programmers haven't stopped programming just because the sales of programming books collapsed and stayed collapsed after the dot-com crash. They haven't just given up on their field and spent the last fifteen years playing on the Xbox. The role that books played in the software industry has been

10. Baldur Bjarnason, "Bridging the Gap: Why Publishing's Future is at Risk," *Publishing Perspectives*, May 8, 2014. https://publishingperspectives.com/2014/05/bridging-the-gap-why-publishings-future-is-at-risk/. Accessed August 9, 2019.

> supplanted by online communities and their byproducts. Sometimes those byproducts are e-books. Most of the time they aren't.[11]

Although a complete investigation of Internet learning systems must await further research on the learn-to-program movement, it is clear that online learning systems are successfully teaching many users how to design and develop their own software. These systems fill an important need in the American economy because there are many who cannot attend traditional college or university courses who hope to benefit from the commercial promise of software development.

But are there also some Internet-based programming tools designed for younger students in schools? To answer this question, let us evaluate the Hour of Code, a contemporary programming movement sponsored by Code.org, a not-for-profit corporation.

Hour of Code

In December 2019, Code.org ran its annual Hour of Code programming event for students around the world. The sessions took place during Computer Science Education Week, held annually to celebrate the birthday of computing pioneer Admiral Grace Murray Hopper (born December 9, 1906). As its name suggests, the Hour of Code is an introductory (1 to 4 hours) coding experience designed to teach the basics of computational logic to those who have never written a computer program. The digital courseware, typically delivered via the World Wide Web, is free to students. The experience is designed to be hosted where teachers and mentors are present and are able to offer additional instruction and support. The Hour of Code is also enhanced with social media and project-sharing technology, to provide students with encouragement, online support, and resources for future learning.

Students and teachers register for the Hour of Code in advance, and then communicate with participants around the globe as the lessons begin and the students start creating projects. The first coding exercises were based on the popular Swedish video game *Minecraft* (now developed by Microsoft Studios). In recent years, coding challenges based on a range of contemporary themes have been added to the list, including characters from the movie franchises *Star Wars* and *Frozen*. The goal of each tutorial is to expose students to logical thinking and basic computer science principles. Learners explore virtual spaces, gather resources, create block structures, and engage in animated activities. The connection to the *Minecraft* platform is particularly effective—not only is the tutorial entertaining, but there is a significant

11. Bjarnason, "Bridging the Gap: Why Publishing's Future is at Risk." Blog at *Publishing Perspectives*. May 8, 2014.

installed base of *Minecraft* users in the age range targeted by Hour of Code advocates. As of May 2019, over 176 million copies of *Minecraft* have been sold across the world on all platforms, making it one of the best-selling video games in history.[12] When people learn to program through *Minecraft* situations, they already have a head start.

Approximately 98,000 Hour of Code events were scheduled around the globe as Computer Science week began in 2019. Code.org co-founder Hadi Partovi appeared several times in the national media, and Apple Computer announced that its customers could register for Hour of Code sessions for free in all Apple stores. Katy Perry, Madonna, Keith Urban, Ciara, and other recording artists hosted "Code Your Own Dance Party" sessions that encouraged teen and tween learners to experiment with coding skills. Beyond the basics, students with more coding experience were encouraged to design new algorithms, build graphics routines, and try their hand at encryption. A graphics-based "AI for Oceans" module taught about machine learning and promoted the ethical use of artificial intelligence.

How effective is the Code.org teaching initiative? In early 2019, the non-profit announced that its network has completed over 720 million introductory programming sessions in its first 5 years of operation, with 46% female and 48% underrepresented minorities using the organization's courseware.[13] At recent Code.org events, 50 U.S. states and most of the largest U.S. cities announced efforts to expand access to computer science education.

The organization publishes a scaffolded curriculum, trains thousands of teachers at workshops, and is now preparing approximately 38% of all U.S. high school students that take the AP Computer Science Principles Exam, with a pass rate of 70%.[14] The non-profit's vision is that every student in every school should have the opportunity to learn computer science, just like the subjects of biology, chemistry, or algebra. The initiative is supported by leading technology companies, industry luminaries, government officials, teachers, and millions of students. What is striking about Code.org is its passion to teach computer programming as part of its computer literacy campaign. This emphasis, using modern, globally-scaled distribution methods, is similar in many ways to much earlier computer literacy movements.

12. Sax Persson, "Celebrating Ten Years of Minecraft," Microsoft Corporation, May 17, 2019. https://news.xbox.com/en-us/2019/05/17/minecraft-ten-years/. Accessed November 10, 2019.

13. Code.org defines underrepresented minorities as "students who are black / African American, Hispanic/Latino/Latina/Latinx, Native American/Alaskan, and Native Hawaiian/Pacific Islanders." See "Code.org 2018 Annual Report," February 12, 2019, 3. https://code.org/files/annual-report-2018.pdf. Accessed August 9, 2019.

14. "Code.org 2018 Annual Report," February 12, 2019, 9. https://code.org/files/annual-report-2018.pdf. Accessed August 9, 2019.

However, the programming technology has been adapted to younger audiences, the organization is more focused on equity and access than earlier movements, and the learning systems have much in common with popular entertainment platforms. In these ways, the learn-to-program movement for children reflects the ethical, cultural, and political concerns that are ascendant in American society.

Campaigns for Computer Literacy

One of my goals in writing this book has been to elevate the importance of computer literacy campaigns and to study how technical movements have sometimes used them to promote the use of software and programming tools to the general public. As a social historian with interests in the history of technology, business, and education, I am curious about how America's computer users learned to write programs in the era of PCs, and what they did with their training and experiences. In the 1970s, 1980s, and 1990s, the diffusion process was supported by the work of skilled author/entrepreneurs who wrote books and magazine articles, and interacted with others at trade shows and user group meetings. The leading computer literacy advocates established technological frames (or discourses) that made computer programming seem important, rewarding, and connected to the emerging practices of digital citizenship and consumer culture in America.

As discussed in Chapter 4, the learn-to-program movement foundered in American schools the mid-1980s, when several influential theorists argued that learning to code was probably less important to students than a comprehensive introduction to computers and society. The topics they preferred included business computing, word processing, database management, and using computers for artistic expression and research. This broad approach to computer literacy allowed districts to pick their own areas of interest, and it prevented computer science from joining the curriculum as an equal to biology, chemistry, physics, or algebra. In addition, a piecemeal approach to computing probably slowed down the rate of computer acquisition in schools. By 1995 there were approximately three computers for every 30 children in American schools, or a 10-to-1 ratio of students to computers.[15]

In 2001, Berkeley education professor Andrea diSessa reviewed the state of K-12 education and argued that real progress on computational literacy could only be made if schools lowered the student-to-computer ratio to 3-to-1.[16] diSessa argued that students needed more immersive computing experiences, and he hoped to bolster the learn-to-program movement by convincing thought leaders that creating

15. Andrea A. diSessa, *Changing Minds: Computers, Learning, and Literacy* (Cambridge, MA: The MIT Press, 2001), 3.

16. diSessa, *Changing Minds*, 4.

software was a new form of social and cultural literacy. In his influential book *Changing Minds: Computers, Learning, and Literacy*, diSessa suggested that programming had the potential to boost not only a student's intelligence but also their ability to learn and solve problems in the future. He co-developed a programming system called Boxer to implement this vision, which provided students with an intuitive way to code using boxes and other visual representations.[17] Boxer was built on the Logo programming language, and it anticipated the visual programming style of commercial products such as Borland's Delphi and Microsoft's Visual Basic (two products introduced in the early 1990s). diSessa's work was well received, and it promoted a new interest in seeing programming as a way of transforming how human beings thought and learned. The Boxer project also provides a link between Logo and more recent visual tools designed for new programmers, such as Blocky, the interface building component used in Hour of Code sessions.

Computer literacy movements provide an intellectual backdrop for the coding experiences that programmers have when they are alone with a computer screen, a keyboard, a compiler, and assorted learning materials. This is a hidden world of coaching and experimentation, where programming authors interact with readers, teachers work with students, and software developers exchange source code and inspiration. One aspect of this exchange is language instruction, style, and syntax, a few of the hallmarks of traditional programming instruction. To investigate this world I have presented several historic computer languages and programming systems. These have included assembly language, COBOL, FORTRAN, ALGOL, Lisp, Logo, classic BASIC, Turbo Pascal, structured BASIC, C, Forth, C++, Visual Basic, and Java. Beyond language and compiler fundamentals, we have also investigated how programmers think and solve problems, how they develop algorithms, how they study and use operating systems, what they learn about computer hardware, how they design a user interface, and how they test and distribute their solutions. There is much more to the world of software development that has not been addressed in this book, but which historians can access through contemporary textbooks, manuals, product documentation, computer periodicals, essays, and the curriculum of schools.

While personal computing has undergone numerous transformations—from early time-sharing systems to mail order microcomputer kits to sophisticated workstations and web servers—the process of learning to code has usually involved an intimate connection *between humans and their devices*. This connection has been

17. The early Boxer programming environment is described in Andrea A. diSessa and Harold Abelson, "Boxer: A Reconstructible Computational Medium," *Communications of the ACM* 29, no. 9 (1986): 859–868.

one of the key features of creating new software for PCs. Lee Felsenstein designed the "Tom Swift Terminal" in 1974 to be a convivial device with replaceable components that the home developer could build, tinker with, and modify. Dian Crayne wrote computer games for IBM PCs in the early 1980s that carefully managed memory and hard disk resources, so that her adventure stories would not outpace the capacities of a typical IBM PC XT. Peter Norton wrote DOS utilities that were intimately connected to, and in control of, the sectors and tracks of single-sided 5.25" floppy disks. Ray Duncan wrote MS-DOS device drivers that enabled new peripherals to work seamlessly with IBM PCs and compatibles. Kathryn Kwinn wrote BASIC and HPL programs that were optimized for the Hewlett-Packard 9826A/9836A systems and its Motorola MC 68000 microprocessor. The Waite Group carefully managed available memory and system resources in the BASIC programs they designed for the new Apple Macintosh. The achievements of these programmers and many more were made possible by an intimate knowledge of PC hardware and software, which they gained through books, magazines, user group meetings, and much experimentation. Having a detailed knowledge of the new platforms was a key attribute of programming PCs, *c.* 1970–1995.

Programming in the Internet age has been characterized by new levels of *abstraction* from underlying hardware and software systems. To web programmers, it is not terribly important what the underlying computer architecture is, what the operating system is, or how memory and system resources are assigned. To learn the fundamentals of Internet programming in recent contexts, software developers have mastered the technologies of HTML, CSS, XML, web browser extensions, cloud-computing services, and security best practices. The traditional programming languages have also been updated so that they are object oriented, platform independent, and implemented through a collaborative, public process. Moreover, the software that Internet programmers create is often stored online "in the cloud" from the beginning of a project to its completion. It only rarely makes an appearance on local media such as CD-ROMs, flash drives, or hard disks. Smart phone app development has further extended this model. Mobile computing programs are often created and hosted in the cloud, downloaded to mobile devices via "App stores," and updated automatically through mobile operating systems.

Despite the differences between traditional programming and Internet programming, however, software development can still be described fundamentally as a process of problem solving with a computer. Programmers define a problem for the computer to solve, write instructions in a given computer language, load the program and its components into computer memory, run the program, correct errors, and distribute the application or tool for others to use and learn from. The historic context of these actions is ever changing, and so are the men and women

who choose to design and construct computer software. As we continue to expand this digital world, and unpack its consequences, it is helpful to consider the many approaches to creating software that have taken place in the past and continue to evolve in the present. The learn-to-program movement and its intersection with personal computing is a vital part of this story.

Author's Biography

Michael J. Halvorson

Michael J. Halvorson, Ph.D., is Benson Chair of Business and Economic History at Pacific Lutheran University, where he teaches courses on the history of business, computing, and technology. He has written widely on European history, application software, and programming personal computers, including the popular series *Microsoft Visual Basic Step by Step*, Pearson (2013). To learn more about the Code Nation project, visit www.thiscodenation.com.

Index

Code Nation

Personal Computing and the Learn to Program Movement in America

Michael J. Halvorson

Code Nation explores the rise of software development as a social, cultural, and technical phenomenon in American history. The movement germinated in government and university labs during the 1950s, gained momentum through corporate and counterculture experiments in the 1960s and 1970s, and became a broad-based computer literacy movement in the 1980s. As personal computing came to the fore, learning to program was transformed by a groundswell of popular enthusiasm, exciting new platforms, and an array of commercial practices that have been further amplified by distributed computing and the Internet. The resulting society can be depicted as a "Code Nation"—a globally-connected world that is saturated with computer technology and enchanted by software and its creation.

Code Nation is a new history of personal computing that emphasizes the technical and business challenges that software developers faced when building applications for CP/M, MS-DOS, UNIX, Microsoft Windows, the Apple Macintosh, and other emerging platforms. It is a popular history of computing that explores the experiences of novice computer users, tinkerers, hackers, and power users, as well as the ideals and aspirations of leading computer scientists, engineers, educators, and entrepreneurs. Computer book and magazine publishers also played important, if overlooked, roles in the diffusion of new technical skills, and this book highlights their creative work and influence.

Code Nation offers a "behind-the-scenes" look at application and operating-system programming practices, the diversity of historic computer languages, the rise of user communities, early attempts to market PC software, and the origins of "enterprise" computing systems. Code samples and over 80 historic photographs support the text. The book concludes with an assessment of contemporary efforts to teach computational thinking to young people.

ABOUT ACM BOOKS

ACM Books is a series of high-quality books published by ACM for the computer science community. ACM Books publications are widely distributed in print and digital formats by major booksellers and are available to libraries and library consortia. Individual ACM members may access ACM Books publications via separate annual subscription.

ISBN 978-1-4503-7757-7

BOOKS.ACM.ORG · WWW.MORGANCLAYPOOLPUBLISHERS.COM