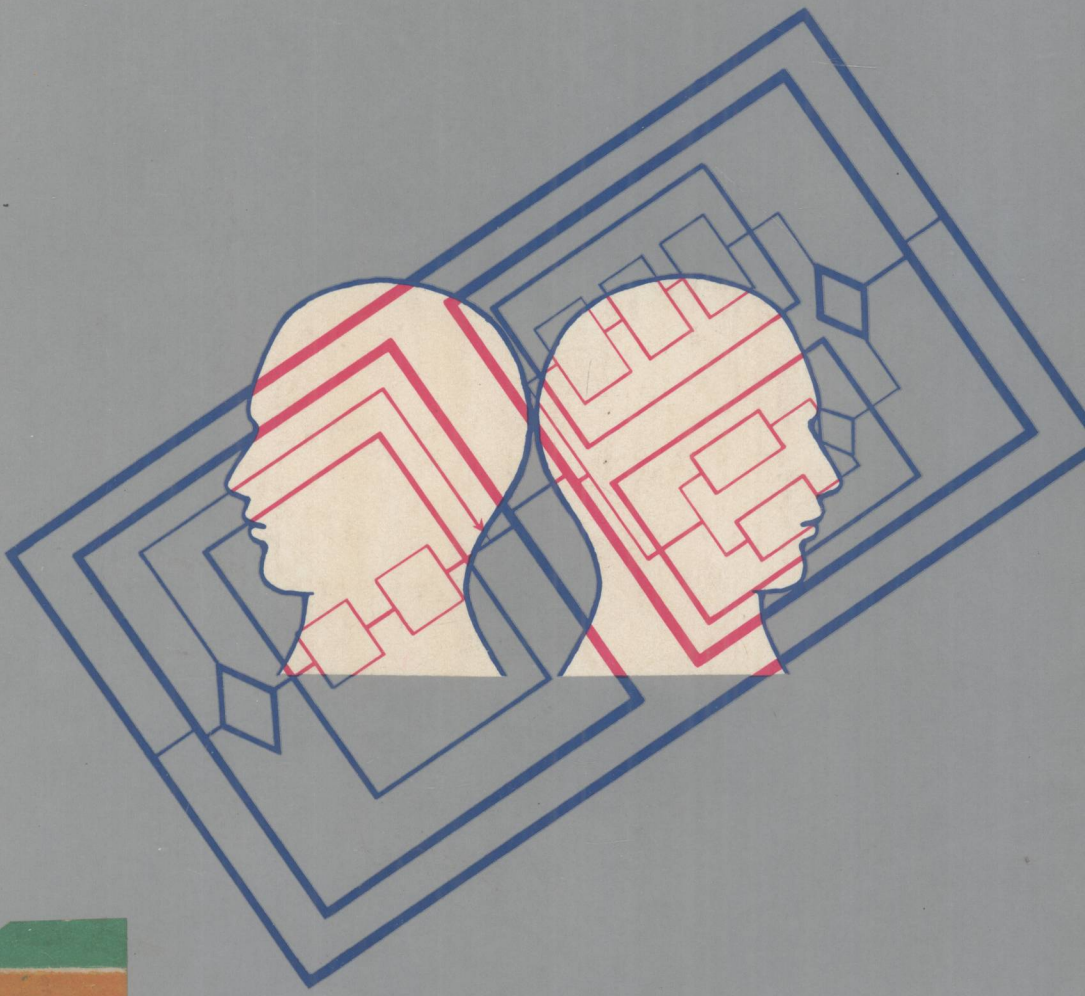
Heidelberg
Science
Library

Programmers and Managers

The Routinization of Computer Programming in the United States

Philip Kraft



Programmers and Managers

The Routinization of Computer Programming in the United States

Springer-Verlag
New York
Heidelberg
Berlin

Philip Kraft
Department of Sociology
State University of New York
at Binghamton
Binghamton, New York 13901

Library of Congress Cataloging in Publication Data

Kraft, Philip.
Programmers and managers.

(Heidelberg science library)
Bibliography: p.
Includes index.
1. Computer programmers—United States. 2. Electronic data processing departments—Personnel management.
3. Industrial relations. I. Title. II. Series.
HD8039.D37K7 331.7'61'0016420973 77-1667

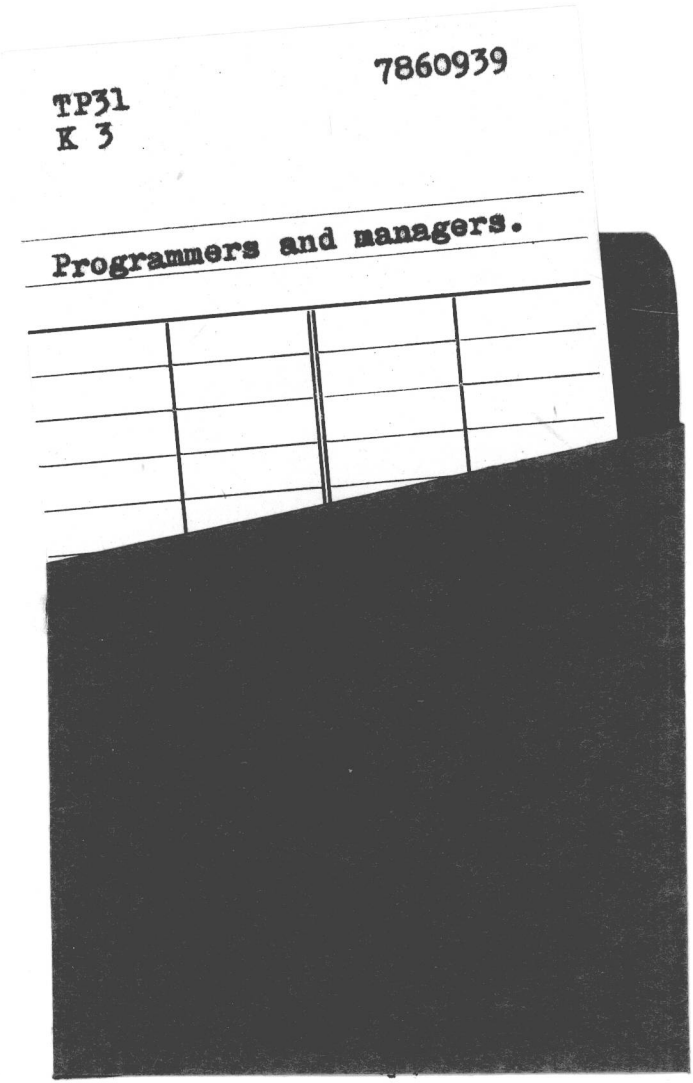
Printed in the United States of America.

9 8 7 6 5 4 3 2 1

ISBN 0-387-90248-1 Springer-Verlag New York
ISBN 3-540-90248-1 Springer-Verlag Berlin Heidelberg

Heidelberg
Science
Library

Philip Kraft



V

W

Y

程序编制者与管理者：美国计算机
程序设计的程序化 F15.70

一九七八年七月廿九日

T

U

Foreword

Norbert Wiener, perhaps better than anyone else, understood the intimate and delicate relationship between control and communication: that messages intended as commands do not necessarily differ from those intended simply as facts. Wiener noted the paradox when the modern computer was hardly more than a laboratory curiosity. Thirty years later, the same paradox is at the heart of a severe identity crisis which confronts computer programmers. Are they primarily members of "management" acting as foremen, whose task it is to ensure that orders emanating from executive suites are faithfully translated into comprehensible messages? Or are they perhaps simply engineers preoccupied with the technical difficulties of relating "software" to "hardware" and vice versa? Are they aware, furthermore, of the degree to which their work—whether as manager or engineer—routinizes the work of others and thereby helps shape the structure of social class relationships?

I doubt that many of us who lived through the first heady and frantic years of software development—at places like the RAND and System Development Corporations—ever took time to think about such questions. The science fiction-like setting of mysterious machines, blinking lights, and torrents of numbers served to awe outsiders who could only marvel at the complexity of it all. We were insiders who constituted a secret society into which only initiates were welcome.

So today I marvel at the boundless audacity of a rank outsider in writing a book like *Programmers and Managers*.

What can be said of this study?

To begin with Dr. Kraft has written a study of computer

programmers unlike any other study of computer programmers. I am simply overwhelmed at what he has managed to learn about the profession, the workplace, and the spirit of computer programming. To my knowledge, this is the first study written from the perspective of programmers themselves. There is, of course, an enormous literature in this area, characteristically written from the perspective of either the hardware (essentially an engineering perspective) or the process to be automated (essentially a management perspective). When Dr. Kraft observes that the limited material available about programmers is not concerned with trying to understand who and what programmers are but rather with developing techniques to control them, he is being accurate in a way which only a genuine "insider" can appreciate. I suspect that countless programmers, reading his account, will indeed experience "the shock of recognizing something which had always been there but hadn't been thought about before."

But this is more than a study of computer programmers written for members of the profession. In providing a clearly stated brief history of the modern computer in terms which all laypeople can readily comprehend, Dr. Kraft has provided an important demystification service in a field which thrives on obfuscation and inflated technical pretense. He has, in addition, provided important insights into the bureaucratic environment within which nearly all programmers must work. Even more significantly, he has also provided insights into the relationships between routinization of work in contemporary industrial society and developing social class relationships.

I suspect this work will be somewhat controversial for many years to come. In the first place, it is clearly not a routine piece of work done in a conventional manner. Since it is not, many professionals in both computer science and sociology may experience considerable difficulty in relating to it. Sociologists who can understand the broader social implications of the phenomena being discussed characteristically know very little about the technology or the minutae of the environment in which computer programmers work. And those who are familiar with the environment either do not know or do not wish to be told about some of the social realities involved.

Dr. Kraft has in a remarkable way fashioned a book which fills the gaps in both fields.

May 1977

Robert Boguslaw
Department of Sociology
Washington University
St. Louis, Missouri

Acknowledgments

Because *Programmers and Managers* itself must be considered a preface to more and better conceived studies of technical occupations and technical workers, I will limit myself here to thanking those who offered help, criticism, and guidance. The people who provided the best of these—programmers, managers, systems analysts, and others who make their living in the software industry—cannot be acknowledged by name. More's the pity, since the debt is all the greater.

Fortunately, I am able to thank directly many people who insisted on being useful, sometimes in spite of doubts or reservations about what I was doing. They are, by accident of alphabetical priority, Cathy Arnst, Robert Boguslaw, Laird Cummings, Pat Dolaway, Daniel Freedman, James Geschwender, Joan Greenbaum, David Gries, Nancy Hall, Mel Leiman, David Noble, James O'Connor, Barry Truchil, and Nancy Zimmet.

The American Sociological Association, through its Committee on Problems of the Discipline, was nice enough to give money to several sociologists to explore issues raised by the study of white collar workers. My colleagues in that undertaking—Theodore Kaplan, John Low-Beer, Martin Oppenheimer, Theodore Reed and Magali Sarfatti-Larson—asked and answered questions, provided reassurance in times of doubt, injected doubts at times of over-assurance, and generally acted like colleagues are supposed to act. I am most grateful to them and to the ASA for allowing us to get together.

The Research Foundation of the State University of New York funded a small preliminary study of programmers through its faculty grants program. Their early support is gratefully acknowledged.

Philip Kraft

Binghamton, New York
January 1977

Contents

Introduction

Programmers, managers, and sociologists

Introductions traditionally are places where authors make one last heroic attempt to justify their work—and to cover their tracks. It is, all things considered, a useful tradition and I too would like to explain how and why this study of computer programmers came to be written.

I live and work near Binghamton, New York. Because IBM is here, because General Electric, Singer-Link, GAF, and dozens of more modest enterprises are here, lots of computer programmers are here too. In the normal course of things, I got to know many of them as neighbors, students, and colleagues. From the start they struck me as marginal people. They did not, for example, fit neatly into the stereotypes that are commonly applied, however unjustly, to engineering workers as a whole. They did not look like engineers are supposed to look (crew-cuts, narrow ties, penny loafers, etc.), nor were they taciturn and awkward around nonprogrammers. They did not appear to be particularly conservative, either politically or socially. Many were women.

But if programmers could "pass" as nonengineers in a nonengineering world, they were not exactly academics or managers or salespeople either. In a word, they were different in ways I couldn't quite define and I got curious. Since my major interests are in the area of work and occupations, a study of computer specialists seemed a natural way of satisfying my curiosity in an orderly and systematic manner.

So I began, in an orderly and systematic manner, to collect information about programmers. I wanted to learn about their history and the history of their field, their education, their occupational backgrounds, distribution, and so on. Almost

immediately I encountered one major problem: hardly any information about these things existed. What little material did exist was of a very peculiar sort. It was obviously intended for managers and personnel directors rather than for the world in general. I don't mean to suggest that there was some sort of deliberate conspiracy involved. It seemed instead to be a peculiar case of an exotic occupation whose obscurity was penetrated only by those in whose interest it was to do so, i.e., managers.

This made me all the more curious, particularly since computer programmers are critical people whose role in data processing is, to put it charitably, little appreciated. It was easy enough to understand why sociologists and other social scientists had, in effect, abandoned programmers to their obscurity. Computers are still relatively new and largely mysterious. Social scientists, even those who make regular use of the hardware, too often are only dimly aware that the machines which do the things they have come to take for granted have to be told what to do by people. Furthermore, it saddens me to say, sociologists cling with most of the rest of the population to a stereotype of engineering and technical workers which extends to computer specialists as well. Engineers, goes the old saw, are dull people. If they don't actually wear brush-cuts and white socks anymore, they remain social Neanderthals. They are "thing-oriented" rather than "people-oriented." They have no interests other than their work, which is, in any case, so esoteric as to be meaningless even to the specialists who do it. And what is perhaps the unkindest cut of all, technical workers are hypocritical political cretins who affect a philosophy of rugged individualism even as they earn a major portion of their incomes from government contracts. They are, in short, nowhere as interesting (or romantic) as the autoworkers or the physicians or streetwalkers that social scientists usually have preferred to study.

By contrast, management researchers have shown considerably more interest. They have compiled an extensive—if, for all intents and purposes, also an underground—literature on computer programmers and other computer workers. More accurately, they have compiled four distinct literatures. The first is a version of the old Norman Vincent Peale/Dale Carnegie brand of moral uplift. Managers are encouraged to develop proper attitudes towards themselves and the programmers who work under them. Their duty is to foster right-thinking, enthusiasm for the job, and loyalty to the company. This is the sort of stuff that gets recited at management conventions and

then reprinted in house journals as inspirational material for up and coming junior executives. Except for its anthropological interest to students of managerial mythology, most of it can be safely ignored.

The second kind has to do with what is usually called psychological profiling. Employers commonly make use of tests which they hope will provide them with clues to the psychological makeup of potential employees. The tests themselves have a number of purposes. One is to predict how well an individual is likely to perform certain kinds of job-related tasks. Programmer aptitude tests are perhaps the best-known examples. Others are concerned with fingering potential "troublemakers," that is, anyone who might disagree with his or her betters or is not likely to adopt pro-company attitudes. In spite of their long history of use by industry, there is considerable disagreement about their accuracy and usefulness and much management literature has been concerned with these tests' relative merits.

A third kind of management writing is of more immediate interest. It consists of the information about programmers' salaries, their distribution by job categories, by industry, and so on. To the extent that this sort of information is accurate it is, of course, very useful. Unfortunately, there is relatively little of it—*Datamation* and *Infosystems* seem to be carrying most of the burden alone—and it applies to something less than the whole population of programmers.

The fourth and last major category is made up of the work of highly experienced programmers who have spent considerable time analyzing the organization of the programming workplace. These writings are fascinating for more than their technical content; the work of F. T. Baker, Harlan Mills, and Gerald M. Weinberg, for example, is also important *politically*. This is a critical and often misunderstood point. The social relations of the workplace are arrangements of people which affect more than just efficiency and productivity. They are also relations of power, of domination and subordination. In the workplace, including the programming workplace, such relations are most clearly expressed in the form of a hierarchy (sometimes referred to as the "chain of command" or "career ladder") and usually represented by formal organizational charts. Mills, Baker, and Weinberg, to the extent they and others like them explore the social relations between various categories of programmers, between programmers and their managers, among programming departments, between programming departments and other departments in the organiza-

tion, are all discussing questions having to do with who is in charge of whom and what. They are, in other words, discussing profoundly political relationships.

Because most managers understand this, the most serious and thoughtful work of management researchers (and their academic counterparts in Schools of Management) focuses on issues which are not technical in nature; they are concerned primarily instead with ways of arranging people to make them amenable to management influence. Discussions of such varied concerns as "dual ladders" (career lines for technical employees), "structured programming," "chief programmer teams," and "egoless" programming (all ways of arranging programmers in particular forms of hierarchy), job structuring, and so on, whatever their technical content, are *primarily discussions of how to manage, not necessarily of how to increase efficiency*.

Put another way, management literature on programmers displays a general concern with developing techniques to get the people managers manage to do what they are told, not simply how to write better programs. It tends to concentrate on ways of figuring out how to predict who can do what kinds of programming jobs ("personnel selection"), how to get programmers to fit into the structure of the organization ("getting on board" or, occasionally, "seeing the Big Picture"), and, all things being equal, to get as much work out of them as possible ("developing motivation" and "acquiring the right attitude").

Finally, it must also be said that much of this management literature is not very flattering to programmers. Some of it, in fact, closely resembles the popular stereotypes of engineering and other technical workers. The major difference is that while such stereotypes have been used to justify social sneers or to ignore programmers altogether, managers have used similar stereotypes to create techniques to advance their own very specific ends. Managers, for example, are quite happy to go along with the popularly held notion that programmers are "thing-oriented" rather than "people-oriented." A common managerial position is that, left to themselves, programmers "would design a system for the computer not for the user." There are endless variations on this theme alone.

What all of this added up to, it seemed to me, was that the limited material about programmers was not concerned with trying to understand who and what programmers were; it was concerned instead with developing techniques to control them. Although managerial wisdom with respect to programmers constituted the bulk of the available material about them,

clearly I was also going to have to look elsewhere for less self-serving information.

Expanding the data base

Partly to balance the obvious managerial orientation of most writings on programmers and partly to gather information that was not readily available, I undertook a series of interviews with many different kinds of programmers—systems programmers, applications programmers, programmers who worked for hardware companies, for software companies, for universities, for hospitals, and for commercial and industrial organizations. I also managed to talk with a very rare species, the middle-aged-to-old programmer, and I also came across a hitherto undiscovered one: the unemployed programmer. For good measure, I talked with managers and with people generally referred to as systems analysts. I also talked with academic people who are training (or trying to figure out how to train) programmers in universities. Finally, where I could, I observed programmers at work.

Whenever I reached some sort of conclusion, I wrote it up and showed it to several of the programmers I had talked with earlier for their comment and criticism. I did this for several reasons. The most obvious was that I wanted to be sure of the facts of a situation and to catch any glaring errors in reporting what I had seen or heard. A second reason was largely a matter of principle. I have never admired the ethics or methodology of social scientists who act as "participant observers" and insert themselves into a community to report on the "natives." I don't approve because the temptation is almost always too great to resist treating the "natives" as objects. If people are good enough to let you bother them with questions and constant hovering around, they have a right to learn what you've learned and to know what you think of them.

But beyond issues of accuracy and principle, returning to the people I had interviewed or observed produced some unanticipated results. For one thing, many programmers who had been the most thoughtful and the most helpful to me hadn't realized just how clearly they had analyzed their own work experiences and personal histories. They had put together, a piece at a time and scattered over one or two interview sessions, a cogent picture of what it was like to be a programmer. When I returned to them with a "story" constructed from their own observations, there was invariably the shock of recognizing something which had always been there but hadn't been thought about before. Recognition prompted more thought, more details, and new insight.

This was obviously a fruitful method for probing beneath