INCOMPREHENSIBLE COMPUTER SYSTEMS:

KNOWLEDGE WITHOUT WISDOM

Ronni Lynne Rosenberg

January, 1980

This research was supported by the MIT Laboratory for Computer Science, with funds supplied by an IBM research contract.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139

This empty page was substituted for a blank page in the original document.

INCOMPREHENSIBLE COMPUTER SYSTEMS:

KNOWLEDGE WITHOUT WISDOM

by

Ronni Lynne Rosenberg

Submitted to the Department of Electrical Engineering and Computer Science

in January, 1980 in partial fulfillment of the requirements for

the degree of Master of Science.

ABSTRACT

An analysis of the incomprehensibility of large, complex computer systems is made. The thesis is that there is strong relationship between system incomprehensibility and the necessity to trust computer systems. A cogent definition of incomprehensibility in computer systems is established, with common themes drawn from interdisciplinary literature dealing with computers and society. Reasons for the creation of incomprehensible computer systems are explored, as well as the consequences (both technical and social) of using and relying on them. The relationship between the real and perceived purposes of computer systems and the appropriateness of trusting these systems is analyzed. Approaches for dealing with the existence of vital computer systems which are functionally incomprehensible are evaluated, and positive suggestions are made.

Name and Title of Thesis Supervisor:

Joseph Weizenbaum
Professor of Electrical Engineering and Computer Science

Key words and phrases: incomprehensible systems, computer system incomprehensibility, technological incomprehensibility, technological systems, societal implications

This empty page was substituted for a blank page in the original document.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

The societal implications of man's interactions with computers is a relatively new area of interest within the computer science community. Because of this, I have felt that a great part of my task is simply to sensitize disparate groups of people to the kinds of problems I present here or, to use a hackneyed phrase, to attempt some consciousness raising. I have been encouraged in my endeavor by the help of many people.

Inspiration for this thesis has been generously and unfailingly provided by my thesis advisor, Joseph Weizenbaum. From the germination of the thesis topic through the research and the actual writing, Joe has been a strong motivating force, helping to sustain me with the very best kinds of intellectual and emotional nourishment. Certainly it is he who has made the experience of producing this thesis such a happy one for me.

More generally, my association with the people who make up MIT's Real Time Systems research group (headed and animated by Steve Ward) has been of enormous benefit to me. They have provided the kind of stimulating, supportive environment in which enjoyable research thrives. Clark Baker has been unfailingly generous with his time in facilitating the computer generation of this document. Eva Tervo has managed to keep things running smoothly for us all. Other friends – Vera Ketelboeter, Carl Seaquist, and Will Fraizer – took the time to read and offer useful comments on the section of Chapter 6 dealing with verification. Jon Doyle volunteered to read the entire document, and his perspicacious comments were extremely useful to me.

Lastly, I wish to avoid pure formality in acknowledging my most deeply rooted debt, to my family, who have offered refuge in trying times and have shared my pleasure in the fruition of this work.

Chapter 1: Introduction

> The technological society contains many parts and specialized activities within a myriad of interconnections. The totality of such interconnections - the relationships of the parts to each other and the parts to the whole - is something which is no longer comprehensible to anyone. In the complexity of this world, people are confronted with extraordinary events and functions that are literally unintelligible to them. They are unable to give an adequate explanation of man made phenomena in their immediate experience. They are unable to form a coherent, rational picture of the whole. Under these circumstances, all persons do and, indeed, must accept a great number of things on faith. They are aware that the major components of complex systems usually work, that other specialists know what they are doing, and that somehow the whole fits together in relatively good adjustment. Their way of understanding, however, is basically religious rather than scientific; only a small portion of one's everyday experience in the technological society can be made scientific. For the rest, everyone is forced to depend upon and have faith in matters about which one has little information or intelligent grasp. It is this condition that Ellul describes as the source of the modern versions of mystery, magic, and the sacred.[1]
>
> Langdon Winner

In recent years, the computer science community has begun to recognize comprehensibility as an important dimension of computer programs. Much earlier, thoughtful observers of the growing preeminence of technology in modern cultures worried about the diminishing ability of people to understand and cope with the technological system with which they were so deeply involved. Today, Dr. Robert Johnson, Vice-President for Engineering of Burroughs Corporation, is not alone in his belief that the most serious problem facing the computer industry is the incomprehensibility of large computer systems.[2] Several factors combine to make computer system incomprehensibility an issue of immediate concern to any society

---

[1] Langdon Winner, *Autonomous Technology* (Cambridge, MA: The MIT Press, 1977), p. 284.

[2] Hearsay.

as committed to technology as our own: notably, the rapid rate of proliferation of new systems, the strong social and economic incentives to use them, and the rising level of dependance on computer systems that have vital, nonreversible impacts on our lives. If widespread unintended consequences of using computer systems that cannot be understood are to be avoided, the problem of incomprehensibility must be analyzed and dealt with now. This thesis is about the incomprehensibility of many computer systems we presently use and on which we depend heavily.

The existing notion of incomprehensibility, as documented in the computer science literature as well as in writings from relevant nontechnical fields (such as sociology and political science), is highly ambiguous; therefore, my first objective, pursued in Chapter 2, is to sharpen this notion and to arrive at a definition from which one may more usefully proceed. Some computer scientists have a tendency to become so involved with the details of the newest methods of rendering computer programs understandable (for example, the technique of structured programming or the avoidance of goto statements) that they seem to forget about more general, higher level issues. Conversely, critics of technology's social role, although they may appreciate the widespread effects of technological innovation, are often hampered by the inability to understand specific technical applications. The study of computer system incomprehensibility thus demands an interdisciplinary approach, based on an understanding of both computers themselves and the technological system of which they are among the foremost representatives. Throughout this thesis, I refer to the observations of diverse groups of people - computer scientists, philosophers, psychologists, sociologists. It is not expected that the reader will be familiar with all the relevant disciplines; therefore, I have provided a "list of characters" - a collection of short biographies of most of the people on whose ideas I have drawn - following the body of the thesis.

In discussing computer comprehensibility, it is useful to first consider comprehension in a broad context as, for example, in communication between people. This need not be a specialized endeavor; most people have a strong intuitive sense of what it means to understand someone else, and it is this informally learned knowledge that one projects onto one's interactions with computers. Sharpening the general concept of comprehension, as it is applicable to human communication, in order to arrive at its much more specific form, namely as it appears in the context of computers, requires competence in computer science. Insights about comprehension, when applied to problem evaluation, system design, programming, etc. (all fundamentally human activities), can help establish what it means to comprehend a computer system and the behavior of computer systems.

Incomprehensibility is not a property of a computer system (in the sense that color is a property of an orange); rather, it is a derived attribute which is dependant upon the context in which a system is used and the criteria according to which it is judged. An airline reservation system might be crystal clear to the reservation clerks who use it every day, but largely mysterious to the systems analysts who attempt to modify it. In addition, incomprehensibility takes on a different meaning in relation to the "front end" of a system (analysis of a problem and design of a system) than it does to the "back end" (utilization and maintenance of a system once it has been implemented). Difficulties are bound to arise either when a system is designed in a haphazard, *ad hoc* fashion (surprisingly common outside the restricted domain of research systems) or when users are forced to communicate with a system without some knowledge of the theory on which its design was based. Indeed, incomprehensible systems often turn out to be those which are based on no well formed theory at all.

It is important at this point to stress the fact that I am concerned with the

incomprehensibility that arises from systems; that is, collections of interrelated and intercommunicating activities, of which computers and computer programs are important, but not exclusive, components. By the term "computer system" I wish to refer to not just computers themselves, but also the people who choose to design, maintain, and use them. Computer systems bring into question much more than just computers and the programs that run on them; for instance, the nature of the problems that we deem suitable for computerized solutions, and the poorly understood processes of problem analysis, system specification and design, and programming.

The kind of incomprehensibility I am interested in does not derive solely, or even mainly, from any easily identifiable errors (such as coding errors or obscure programming), but from more elusive problems with the way we think about and deal with technology in general, and computers in particular. If we apply computer technology inappropriately or indiscriminately (for instance, if we are more motivated by an eagerness to make use of computers than by the actual effectiveness of applying computers in a given application), we may end up having difficulty understanding the relationship between the original problem and the computer system constructed in response to it. In some cases, "problems" are artificially created or tailored to make them better suited for applications of current technology. Computer systems that arise from such situations can be functionally incomprehensible - incomprehensible in relation to the problem that a system's users believe it is "solving."

Already in the present discussion, I have turned to the question of how does incomprehensibility arise in a computer system. In Chapter 3, I examine factors which can lead to the generation of incomprehensible computer systems. Concerns about the process of programming are relevant here: Gerald Weinberg has reminded us that programming is a human activity with a psychological component which is often ignored, but which significantly affects the quality of programs which

are written. The activity of programming a large computer system is plagued by what some people have described as problems of communication. Interactions between the diverse groups of people who are touched by computers can be stifled by the elitist belief, held by some specialists, that non-technical knowledge is not very relevant in the design of a computer system, or that outsiders should not question the appropriateness of applications of technology. Professional isolationism helps distort the knowledge that coders, system designers, users, and everyone in-between can have of a system (by coders, I refer to people who carry out fairly routine programming jobs which are handed over to them by other people who are more involved with higher level jobs like the design of a system). Bizarre stereotypes of programmers as solitary individuals who are detached from other people are reinforced by the informal attitudinal training of software workers, which tends to discourage curiosity beyond the level of specific, unconnected programming tasks.

In addition to relatively low level problems which may be inherent in the programming process, there are broader, societal issues which have important impacts on our relationship to computation; these issues are discussed in Chapter 4. As I mentioned before, a thorough consideration of the problems entrained by incomprehensible computer systems requires both an understanding of computers and a high degree of sensitivity to the social contexts in which computers play an important role. In the modern world, one ought not talk about social "problems" without talking about technology, nor discuss technology without taking into account its social context.

The present organization of society is such that there is an air of inevitability about the role of technology. Our ability to critically evaluate social problems and proposed computer solutions to these problems is strongly influenced by what

appears to some as our acceptance of the autonomy of technology at the expense of our own human autonomy. The perception of technology as an irresistible force leads to a situation where the usefulness of the computer is often assumed, even when a given application of computer technology is by many criteria inappropriate; where the widespread use of computers is accepted in spite of the fears and misgivings of many people; where the surface appeal of quick technological fixes for pressing problems often causes a redefinition of our problems to make them more amenable to computerized solutions. Present day society uncritically accepts a way of life founded on technical necessity and shaped primarily by what Langdon Winner calls the technical (rational, artificial, productive) mode of activity and thought.[3] Our love affair with technology has always been characterized by blindness, and the so-called computer revolution is only the most recent example of this technological preoccupation.

Chapter 5 of this thesis is a discussion of the results of the use of incomprehensible computer systems and of the social system in which they are embedded. Some technical results are the inability to ensure adequate reliability for many large, complex computer systems, and the resistance of of these systems to even minor modifications. Other consequences are extensions of points I raise in my discussion of potential sources of incomprehensibility; for instance, the obscuration of the root of a problem as a result of over-rationalizing the processes of evaluating problems and of planning solutions to them. If a computer application is viewed solely in information processing terms, there may be a gap sensed between the problem and the computer system which was designed to solve it, but which in actuality attacks only those symptoms of the problem which were easy to translate

---

[3]Winner, p. 127.

into the form of a computer program. The size of this gap reflects the importance of those aspects of the problem which could not be expressed according to reductionist criteria, and hence were never reckoned into the design of the computer system. Technological elitism, which I cited earlier as a source of incomprehensibility, is strengthened by the existence of computer systems which only a few technically trained people can plausibly claim to understand or be able to maintain. In Kenneth Laudon's words, we are experiencing " . . . a legitimization of technological 'experts' at the expense of poets."[4]

On a still broader level, an attempt to calculate the costs of living in a technologically based society must include some estimate of the personal price that is paid - the human suffering. There has been a gradual adaptation of human needs, desires, and thought processes to that which contemporary technology can explain and satisfy. What happens to the self image of people when "all the business of life, from work and amusement to love and death, is seen from the technical point of view"?[5] As a result of society's continuing quest for the perfection of the machine, some people now seriously question whether we will be able to keep pace with our computers. I will examine these and other unintended, but perhaps unavoidable byproducts of the computer "revolution."

Perhaps the most serious effect of the widespread use of computer systems is our growing dependance on computer systems that we do not understand. The use of computer systems that are not comprehensible can result in the loss of our control over the processes that computers monitor, as we become increasingly

---

[4]Kenneth Laudon, review of *The Conquest of Will: Information Processing in Human Affairs*, by Abbe Mowshowitz, *Science*, 193 (September, 1976), 1111.

[5]Jacques Ellul, *The Technological Society* (New York: Vintage Books, 1964), p. 117.

dependant on the information, and even the decisions, which are the output of the computers. In actual applications involving large, complex computer systems, people using such systems' output often come to rely on "what the machine says." They are effectively in a position of either having to do the work the computer is supposed to be doing, or of having to uncritically accept the output of the computer. The inherent complexity of many systems makes it almost impossible to adequately explain, predict, or trace their operations. Nevertheless, in using these computer systems, one must cross a threshold from reliance to unjustified trust. Trust is deeply embedded with human values, and it is not at all clear that the projection of these values onto man-machine interactions is appropriate or desirable.

An unwillingness to depend on computers to aid in important decision making may be deemed evidence of an "antitechnological" attitude. Nowadays, an attitude of technological optimism and trust appears not to need justification, while technological distrust is often greeted with hostility. This social pressure against criticism of technology is not irrelevant to my discussion of incomprehensibility.

I believe that there is a correspondence between our trust in computer systems and the purposes of the systems (both our understanding of these purposes and how well the systems actually fulfill them), that there is a discrepancy between the real and the perceived purposes of many computer systems, and that there is a strong relationship between our comprehension of these purposes and the appropriateness of trusting the systems. It is our trust in computer systems which makes us so vulnerable to their effects.

One direct and dangerous effect of trusting incomprehensible systems is the erosion of the meaning of responsibility for one's actions. Who is to be held responsible for modern computer systems that have literally evolved into their present forms and that simply cannot be said to have authors? Already, some

people are suggesting that the computer be responsible for its share of the decision making load, but it is far from clear what possible meaning computer responsibility could have for the people affected by machine made decisions. Some advocate the idea that systems retain control over ongoing processes. In practice, incomprehensible systems become autonomous, unchallengeable authorities to whom a society of users abdicate responsibility. According to Joseph Weizenbaum, " . . . responsibility has altogether evaporated. No human is any longer responsible for 'what the machine says.' Thus there can be neither right nor wrong, no question of justice, no theory with which one can agree or disagree, and finally no basis on which one can challenge 'what the machine says.'"[6]

Certainly I do not wish to leave the reader with nothing but dismal pronouncements about the impossibility of individual action against an autonomous technology. In Chapter 6, I discuss alternate courses of action in the face of incomprehensible computer systems. First, I consider various technical "solutions." These include verification proofs, reliability studies, and modern programming practices like structured programming. In the course of analyzing these and other means of injecting understandability into computer systems, it becomes clear that, for the most part, they attack only the incomprehensibility of computer *programs*. However, the enforcement of structure on the product (the program) does not necessarily enforce structure or comprehensibility on the process which created it (the design of a system), and it is the larger computer system that is the concern of this thesis.

I believe that system incomprehensibility is fundamentally not a problem in the engineering sense of that word, and that the most interesting kinds of

---

[6]Joseph Weizenbaum, "On the Impact of the Computer on Society," *Science*, 176 (May 12, 1972), 613.

incomprehensibility are not to be dealt with by technical means. Rather, we must make a conscious effort to widen our perspective beyond a narrow professional one. The social consequences of using incomprehensible computer systems have direct impacts on the meaning of responsibility, the ethical and moral burdens of computer scientists, and the self image of all people. It is necessary to reckon not only the technical, but also the human costs of the technological system; in Lewis Mumford's words, to ask "not what is good for science or technology, . . . , but what is good for man . . . ."[7]

There are positive steps which can be taken to avoid the potentially dangerous effects of the existence of vital computer systems that are functionally incomprehensible. As Vice-Admiral H. G. Rickover has pointed out, a good beginning may be made by reflecting on whether or not everything hailed as progress actually contributes to happiness (or whatever else the reader believes sustains human culture), and remembering that we alone are responsible for our technology.[8]

---

[7]Lewis Mumford, "Authoritarian and Democratic Technics," in *Technology and Culture*, ed. by Melvin Kranzberg and William H. Davenport (New York: Schocken Books, 1972), p. 58.

[8]H. G. Rickover, "A Humanistic Technology," in *Technology and Society*, ed. by Noel deNevers (USA: Addison-Wesley Publishing Company, 1972), pp. 22-23.

Chapter 2: Understanding Computer Systems

> There can be no total understanding and no absolutely reliable test of understanding.[1]
>
> Joseph Weizenbaum

> I understand[2]
>
> ELIZA

It is acknowledged within the computer science community that the incomprehensibility of computer systems is a major problem associated with the rapid proliferation of large scale computer applications. Nevertheless, incomprehensibility has proved to be a most difficult concept for both computer scientists and social scientists (working from technical and non-technical perspectives) to define. Computer specialists hint at the issue of incomprehensibility when they discuss the "software problem," which is really a whole collection of problems that make computer programs, particularly large ones, intractable. There seems to be something inherent in large computer systems, perhaps it is the complexity of such systems, which fosters incomprehensibility. A large computer system is worked on by so many people over such a long period of time that there is finally no group of people who can be said to be its authors or who understand it in any useful sense; for example, well enough to guarantee its reliability.

It is probably premature to jump right into a discussion of computer incomprehensibility before saying something about comprehension in the context in which we are most familiar with it – that of interpersonal communication. Our feelings of understanding another person and of being understood by others are not

---

[1] Joseph Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation* (New York: W. H. Freeman, 1976), p. 193.

[2] A computer program, published in 1966, which makes possible certain kinds of natural language conversation between man and computer.

formally learned, but are instead the results of intuition. Some aspects of this intuitive knowledge seem to be shared among all people by means of unspoken channels; communication is by definition an act of sharing.[3] The assumptions which make human communication work (to the extent that it does work) most often remain unstated. The incomprehensibility of computer systems is such an elusive notion because our sense of what it means to understand is not externalized when we change our frame of reference from that of people to that of machines, and begin to talk about understanding computers.

2.1: Interpersonal Understanding

As a result of our interactions with other people, we arrive at an informal definition of understanding, according to which understanding is a function not only of words that are spoken (or thoughts that are communicated in other ways), but most importantly of the speaker and the listener, who bring something of themselves to any exchange with another person. Comprehension, as implicitly defined in human relationships, is to a large extent founded on shared experiences and values. Interactions between people nearly always require some element of faith, based on our trust that the other person will "know what I mean." This trust is justified only because "all people have some common formative experiences . . . . There is consequently some basis of understanding between any two humans simply because they are human."[4] We find that our trust is best rewarded by people who see the world as we do, and in particular by those people who have had experi-

---

[3]The Oxford English Dictionary (Oxford: Oxford University Press, 1933), p. 699.

[4]Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation*, pp. 192-193.

ences similar to ones that we have had and that were important to us. An understanding between people is not something carelessly acknowledged. It is only when we feel that someone else's interpretation of the world is sympathetic to our own that we are ready to believe that that person can understand us.

One of the most important observations that can be made about human understanding is that there are insurmountable limits to the level of communication that can be achieved.

> Since, in the last analysis, each of our lives is unique, there is a limit to what we can bring another person to understand. There is an ultimate privacy about each of us that absolutely precludes full communication of any of our ideas to the universe outside ourselves and which thus isolates each one of us from every other noetic object in the world . . . .
>
> To know with certainty that a person understood what has been said to him is to perceive his entire belief structure and *that* is equivalent to sharing his entire life experience.[5]

Attempts to communicate with other people are largely acts of faith, substantiated in part, though never entirely, by a common language, social background, environment, experiences, etc. I wish to stress this point - the significance of a common humanity in human understanding - because it is precisely this common humanity that is not part of our "relationships" with computers, and thus cannot play any role in an understanding between man and machine.

2.2: Comprehending Computers

An emphasis on the human element of interpersonal communication leads directly to the belief that understanding must take on a very different meaning in relation to interactions between men and machines than in relation to interactions between people. Just what it is that characterizes this difference is not immedi-

---

[5] *Ibid.*, p. 193.

ately apparent, since there are a number of things that we can mean when we say we understand a computer system. The nature of our understanding of computers depends on the context in which we make use of them (which may be as a neutral storehouse of information or as a decision-maker on which we depend heavily) and varies in relation to the many different elements that combine to form a computer system (from computers themselves and computer programs to the human beings who interact with them, and even the stated and the perceived goals of a system).

2.2.1: Program Output

The simplest interpretation of understanding in the context of computers is that of understanding the output of a computer. One means of achieving this level of comprehension would be tracing the operations of a computer program(s) that generates particular output that we are interested in and wish to understand. Unfortunately, there are strong pressures that can effectively preclude such direct involvement in computer applications.

There are numerous social factors which subtly encourage the use of computers and discourage any serious criticism of them. One aspect of the "American way" of life is the use of the latest technological "breakthroughs"; Norbert Wiener gives a compelling discussion of what he terms "gadget worshiping":

> Of the devoted priests of power, there are many who regard with impatience the limitations of mankind, and in particular the limitation consisting in man's undependability and unpredictability . . . .
>
> In addition to the motive which the gadget worshiper finds for his admiration of the machine in its freedom from the human limitations of speed and accuracy, there is one motive which it is harder to establish in any concrete case, but which must play a very considerable role nevertheless. It is the desire to avoid the personal responsibility for a dangerous or disastrous decision by placing the responsibility elsewhere: on chance, on human superiors and their policies which one cannot question, or

> on a mechanical device which one cannot fully understand but which has a presumed objectivity . . . .
>
> Once such a master becomes aware that some of the supposedly human functions of his slaves may be transferred to machines, he is delighted. At last he has found the new subordinate - efficient, subservient, dependable in his action, never talking back, swift, and not demanding a single thought of personal consideration.[6]

Today, gadget worshiping is often implicit in the use of large computerized information systems. In the minds of many people and perhaps of society as a whole, an unwillingness to make use of current technology bespeaks a backwards attitude and even a lack of support for good old American ingenuity. People who suggest checking up on computers (which surprisingly many people still think of as the machines that never make mistakes) are thought of as simply not keeping up with the times, and because of this they may not be deemed suitable for professional advancement. Few people can be expected to stand up to the social pressures exerted by the workplace (in the form of professional recognition) and by a society in which the omnipresence of technology is almost completely accepted.

Wholly apart from the ways in which a computer system may directly and tangibly benefit a company, computerizing one's business operation is a status enhancing act; computers are image builders. It is not difficult to find serious businessmen (and even more frighteningly, serious computer science researchers) who use computers at least to some extent simply because everyone else uses them, even though this justification for employing computers reduces to nothing more than the age old syndrome of "keeping up with the Joneses." In a recent public talk, a bank executive admitted that some of the portfolio counselors in his department display computer output primarily to impress clients by enhancing the image of the

---

[6]Norbert Wiener, *God and Golem, Inc.* (Cambridge, MA: The MIT Press, 1964), pp. 53-55.

bank.[7] In some cases, it is not much more than such insubstantial reasons that underlies the large investments of time and money needed to build a computerized business system. Given the predominantly social role that some systems fulfill, it is not surprising that understandability merits a low priority in these situations.

In addition to social considerations, there are strong economic factors that have a major influence on a business's use of computers. The pressures of economic competition can create a situation that is far more dangerous than mere wastefulness suggests. The real danger lies in the fact that constraints of time and money prevent most users of computer generated output from verifying the correctness of the output that they must often depend upon in making decisions. John Kemeny, president of Dartmouth College, remarks that "It is a simple economic calculation that a man who earns $25,000 a year cannot afford to spend a week doing by hand something that a computer can do in five minutes."[8] Unfortunately, the opposite may be true. Particularly in circumstances where vital decisions are being made (*e.g.*, military command and control applications), what we cannot afford is an unjustified dependance on information and decisions output by a computer. Nevertheless, given the existing emphasis on cost-effectiveness in the business world, understanding computer systems by directly monitoring their output is frequently not feasible.

---

[7]Laurence Reineman (Vice-President, First National Bank, Boston, Massachusetts), "Computers and the Workplace," lecture sponsored by MIT Program for Science, Technology, and Society, March 7, 1979.

[8]John Kemeny, *Man and the Computer* (New York: Charles Scribner's Sons, 1972), p. 107.

2.2.2: Theory of Behavior

A more realistic way of satisfying ourselves that we understand a computer system (and the same way in which psychologists often satisfy themselves that they understand some aspect of human beings) is to have a theory of the behavior of the system, to which we can turn in order to explain and to verify the output of the system. Such a theory can prove extremely useful in understanding the operation of a computer system and not just the individual programs that comprise it. This higher level of explanation is more economical than a detailed account of computer programs (perhaps in the form of commented program code or lengthy documentation), and it allows us to predict the output of a program that is the implementation of the theory and to establish when that program malfunctions. It will still be possible for a sufficiently complex system to surprise us, but this need not invalidate our theory; rather, non-standard system behavior may act as a test of the validity of a theory, and the theory itself can serve as a check of the correctness of the system.

There are certain trade-offs exacted in return for the convenience and the security of having a well founded theoretical explanation of a computer system. In defining our understanding of a computer system by our understanding of the theory behind the system, we are enforcing some distance between the system and its users. People who make use of a computer system may come to think of it more in terms of the theory that constitutes a behavioral abstraction of the system than of the underlying hardware and software. Thus, although the existence of a theoretical foundation may make clear when a computer system does something wrong, it will require a shift in perspective to know how to modify the underlying code to prevent future occurrences of the aberrant behavior of the system. The code itself may be largely incomprehensible, so that modifications may very well damage

system reliability. In any case where a system "bug"[9] is discovered, but either its source persists in being elusive or the costs of modifying the system are too great, users of the system are left with a choice of abandoning the system or adapting their behavior to the system's quirks. All too often, people come to depend on a computer system too much to consider working without it and reverting to older, pre-computer ways of doing things. Thus, it is not unusual to see an adjustment of human needs so that they may better correspond to that with which existing computer systems can reliably (which is to say, without much intervention on our part) provide us.

When a computer system is described in terms of a theory of its use, the problem of comprehending the system can be viewed as a collection of problems that involves understanding first the theory itself and then the relationship between the theory and the system.[10] The most easily comprehended systems are those that deal with some widely understood, unambiguous subject matter. An example of a system that is extremely large and complex, but nevertheless understandable in these terms, is the MACSYMA computer system for solving problems in the integral calculus. The mathematical theory on which this system is based is sufficiently well defined that it is not open to much dispute nor subject to major modifications. If MACSYMA makes a "mistake," the user should be able to detect

---

[9] An unwanted and unintended property of a program.

[10] In other words, the system is a model of the theory: " . . . if we view the theory we incorporate into a program as an *uninterpreted* theory, we are free to view the computer's behavior as satisfying one interpretation of the theory, so that any programmed computer can be viewed as instantiating (on one interpretation) the theory incorporated in its program. The tokens of computer behavior that on one interpretation are uttered *descriptions* of the behavior of some other entity instantiating the theory, can on another interpretation be viewed as themselves instances of behavior predicted by the theory." Daniel C. Dennett, *Brainstorms - Philosophical Essays on Mind and Psychology*, (USA: Bradford Books, 1978), p. 194n.

the error by referring to this underlying theory. Computer system users need some knowledge of the theory according to which a system has been constructed in order to avoid a situation where a system is set up as an unquestionable authority and to justify the trust that is implicit in many interactions between computers and people.

Once we have accepted the validity of a theory, there is another aspect of understanding a computer system: we must convince ourselves that the system is a valid implementation of the theory. Ideally (if our understanding of the theory is nearly complete), we can subject a system to a number of sample cases designed to exhaustively test it. If a system fails to crack under the stress of extensive, worst-case testing (assuming we can always implement such testing), our confidence in it will be at least partially established. None of this, however, touches the question of when is it appropriate for us to rely on a computer system. The appropriateness of trusting a system in a given application depends on much more than even the most thorough testing; I will have more to say about this issue later in this paper.

The reason most often given to justify the need for computers is the complexity of present day society - the immense quantity of information that must be processed and the variety of connections between data that must be stored and analyzed. If a theory captures the essential means and ends of a problem we set out to solve with the aid of computers, then expanding our view of a computer system from the level of its actual components (such as physical devices and software) to that of the theory on which it is based can be a great aid in coping with the complexity of many modern systems. Enormously large, complicated systems like MACSYMA deal with complexity by giving up local understanding of the details of the system, while maintaining a more global understanding of the goals of

the system and how it goes about achieving them.

We may not be able to guarantee high level understanding of a system if it is not founded on robust theories; however, the existence of a well founded "theory of use" can enable us to comprehend a large computer system without having to keep track of the numerous details of its operation. Most of MACSYMA's users can understand the system well enough for their purposes without being aware of the software that interprets and carries out their requests. We may relinquish direct understanding of many details of a computer system to accomplish very large or complex tasks that are better understood in terms of a concise theoretical explanation than in terms of a series of computer programs. Some computer applications are simply too large in scope to be usefully understood in detail by anyone. It is important to note that the difficulties raised by the complexity of such applications are surmounted more by our own understanding of the problem domain (which may be in the form of a well constructed theory) than by the particular computer system that may implement a solution to the problem.

2.2.3: Criteria and Context of Use

Our understanding of any computer system depends on the context in which we make use of it and the criteria according to which we judge it. We may comprehend a particular system well enough to feel confident of its reliability in normal usage, but not well enough to risk making changes to it. It seems reasonable to expect the computer systems we use to demonstrate high standards of reliability, maintainability, flexibility, modifiability, etc. Many systems satisfy some of these criteria, but it is questionable how many systems achieve enough of them to really satisfy their users or to justify their continued use.

It is the job of system designers to decide which characteristics will be

most vital to the operation of a given system, and to make clear the extent to which certain criteria will be satisfied by the system. It is then up to a system's users to be aware of the strengths and weakness of the system (just as we accept limitations in people), and not to attempt to interact with the system in ways that are contrary to its "character." In other words, people who deal in any context with computer systems should consciously decide in what ways it is appropriate for them to interact with a computer. For instance, it seems obvious that a system that is not designed to reliably support modifications (such as the federal social security computer system) should probably not be modified once it is in operation.

Unfortunately, simplistic guidelines like the one mentioned above are not likely to be of much use in the complicated environment that surrounds a large computer system. To begin with, modifiability and other important properties of any computer system are in general not "built into" a system in the process of system design or implementation.[11] In practice, the major influences on the design of a computer system often turn out to be the cost of the system and externally imposed deadlines for its completion. Computer professionals must at times settle for getting a system up and running with the smallest respectable amount of testing and documentation. Attributes like reliability may not be investigated until a system is in use and, even worse, until people have begun to depend on it; at this point, it may prove necessary to make changes to a system whose reliability in the face of modifications is highly questionable.

It is often the case that the extent to which a computer system exhibits

---

[11] An important exception to this statement is the following: programmers and system designers often leave "hooks" in their programs, so that foreseen extensions can be made more easily and more reliably in the future. However, one's vision of what hooks should be left is often severely limited.

such features as modifiability and reliability is revealed while the system is in use, and that our understanding of a system is formed by means of "experiments" over the course of time. Since the consequences of computer systems can be both non-trivial and irreversible, it is of the utmost importance for us to consider who are the real experimental animals. Clearly, it is not always possible or advisable to determine a system's correctness or resistance to change by, for instance, implementing some crucial modifications and watching the results of the changes on the actual operation of the system (as opposed to a test run).

The program code of commercial systems can easily develop into a patchwork of quick fixes for unforeseen difficulties. Clever but obscure technical patches may be hastily applied to the original programs in response to pressures from corporate management for minimal production time and system cost, and from users for minimal system down time. Tracking down a difficult "bug" can lead to insights which, if perceived before the system was implemented, would have resulted in a different system organization than the existing one. However, once a system is in operation (in fact, once its implementation is under way), restructuring it according to new knowledge is usually not feasible. Besides the obvious economic factors which effectively preclude serious reorganization, there are psychological influences: after the interesting, challenging work of locating a bug is finished, the job of restructuring a system may appear comparatively tedious. Thus, even if there is no pressure to work quickly, the temptation to patch up a system, rather than to attempt to restructure it, is strong. Unfortunately, it is frequently impossible to predict the behavior of a computer system that consists of a collection of patched-together programs.

Particularly in the case of systems with unclear or potentially dangerous, nonreversible consequences, issues like reliability should be analyzed well before

the system is put together. Such an analysis should be made from a non-technical viewpoint as well as from the standard technical viewpoint according to which some computer systems are already judged. A given reliability factor may warrant the use of one system in an academic application, but not the use of another system as an air traffic controller, and the methods of making these judgments are not exclusively those of a formal technical study.

The most thorough analysis of a computer system would have to take into account subtle issues, such as the degree to which users of a system are likely to depend on it and the possible consequences of this dependance. Issues like these, which bring into play the people who use, depend on, and are otherwise affected by computers, are likely to be more inscrutable than computer systems themselves. Although there has been recent evidence within academic circles of sensitivity to the "human factors" of computer systems, few critical, non-technical sentiments have filtered through enough economic and social channels to be evident in the design and use of existing computer systems.

Our knowledge of the extent to which a computer system measures up to certain important standards (which will in all probability be tested at some point in the course of its lifetime) is a major factor in our understanding of that system. A lack of attention to criteria such as those I have talked about here will severely limit the ways in which we can reliably, comprehensibly interact with computer systems. Shallow levels of understanding are hardly enough to warrant the high degree of trust that many people place in computers. The advisability of exercising great caution in constructing and using bigger and better computer systems seems obvious (particularly in the unexampled, poorly understood areas of some current computer science research), but some of the most highly respected computer scientists proceed in their work largely by employment of the method of trial

and error, without much consideration of the ramifications of their work.

As a result of the previous discussion, we can see that the criteria by which we judge a computer system are closely related to the context in which we use it. Different, and probably less stringent criteria would be appropriate in the consideration of an airline reservation system than of a military command and control system. Context and criteria of use help determine the risks and benefits of using a computer system - what might be called the system's "degree of vitality" - on which any discussion of the appropriateness of utilizing the system should be based. Ultimately, a computer system's degree of vitality depends on the ways in which the system may affect people.

Finally, the context of our use of a computer system has something to do with our perception of the relationship between the system and the problem it purports to solve - the extent to which a computer system attacks the problem we actually want it to solve. As I will discuss later in this thesis, computer systems designed to solve social "problems" (for example, the failure of most modern school systems to educate or to motivate their students) often miss their mark. Many times, this is so not because of poor programming, but because we do not understand the nature of the problem. In applying computers to a relatively superficial symptom (such as the current shortage of teachers) of a persistent social problem, rather than coming to grips with the real source of the difficulty (which may or may not be "computable"[12]), we are taking the easy way out of the problem. We may understand how our system deals with the problem symptoms on which we have concentrated our attention, but we will probably not understand why the system

---

[12]By "computable," I do not mean to refer to any technical definition of what can and what cannot be represented by a computer algorithm, but rather to a more intuitive sense of the kinds of problems to which a computer system may *appropriately* be applied.

does not solve the original problem (and it is likely that it will not solve it), since we never acknowledged the root of the problem - the human sources of the difficulty - in the first place. The utilization of computers as "patch" solutions to poorly understood problems can be the source of a subtle kind of incomprehensibility that is extremely difficult to detect. The incomprehensibility of such systems derives from the absence of a well thought out explanation of the difficulty at hand, preceding the design of a computer system to resolve that difficulty. When a system is constructed to model or otherwise refer to a human activity, its design should reflect first and foremost an understanding of that activity and not just an attempt to operationally replicate it.

2.2.4: Front End Versus Back End

In discussing computer systems, we must make clear whether our viewpoint is that of the designers of a system or of its users. Understanding takes on a different meaning with respect to the design and generation of a computer system (what I will refer to as the "front end" of the system) than it does with respect to the use of a system (the "back end").

A system's front end comes into being before the system is even built, with the basic decision to use computers to deal with a particular problem. Before the technical design of the system can begin, an understanding of the problem must be arrived at. Often, the people who have the most complete knowledge of the problem domain are not the same people who set up a computer system in that domain. Thus, in addition to difficulties inherent in the problem being analyzed, there is likely to be another stumbling block - that of communication between the technical specialists who set up the computer system and the non-technical workers who provide input into the specifications of the system and who will eventually make use of it.

An obvious way of dealing with this communication predicament is to include both computer engineers and system users in the design of a system. Unfortunately, there is often considerable difficulty in bridging the gap between different perspectives of a computer system (that of the engineers and that of the users, for instance). This difficulty cannot necessarily be resolved technically (for example, by developing a sufficiently high level programming language that serves the needs of the various groups of people who use a computer system), since it arises, at least in part, from discrepancies in the ways in which different people interact with and think about computers.

Besides difficulties with communication, there are other difficulties that can preclude a well organized front end of a system. Consider the fact that computers are often introduced into situations which have never been overtly organized along rational lines. With the suggestion of computerized operations, an existing, informal system must be converted into a form that is suitable for representation by computer programs. In applications outside the realm of academic research (*i.e.*, applications that are often characterized to a larger extent by "sloppy" human interactions than by well defined rules), the task of rationalizing the work often proceeds in a haphazard, *ad hoc* fashion. Moreover, it is often the case that the work itself is redefined in terms of the tools (in the present discussion, computers) that are available.

One case study which exemplifies the difficulty of converting pre-computer activities into computerized systems is the design and implementation of the Bank of America computer system in the late 1950's. The designers of this system quickly discovered that there was no organized system of rules and regulations that completely governed the activity of banking. Despite the large scale and the obvious complexity of this system, transactions that were not strictly routine were

often handled informally, many on a case-by-case basis. In this way, bank personnel were able to take into account unusual aspects of a client's situation and, to some extent, personalize their decisions. As the banking system grew to large proportions, no formal policy was ever necessary because the system worked. Because life gives feedback, contradictions can be resolved informally (sometimes we call this leniency).

It may not always be in the best interests of a system to formalize its decision making processes. In the case of the Bank of America system, the decision to utilize a computer system necessitated the establishment of a system of rules that explained banking procedures; as a result, the banking system was changed. Personalized decision making was discouraged as a result of the introduction of computerized banking. Contrary to Vice-Admiral H. G. Rickover's claim that "of technology it can be truly said that it is not 'either good or bad, but thinking makes it so,'"[13] the computer did not remain neutral with respect to the activity of banking. The presence of the tool substantively changed the activity that was computerized, even though this was not the original intention.

Haphazardness in the organization of the front end of a computer system is bound to propagate to the back end, as a result, a system's users will experience varying degrees of difficulty and confusion in utilizing the system. When a computer system is introduced to replace an activity that makes significant use of non-standardized channels of communication, important characteristics of that activity are liable to be lost in the process of converting it into a computable form. Users may find that different decisions are being made than had previously been the

---

[13]Vice-Admiral H. G. Rickover, "A Humanistic Technology," in *Technology and Society*, ed. by Noel de Nevers (USA: Addison-Wesley Publishing Company, 1972), p. 21.

case, that seemingly small mistakes can have nontrivial, unforeseen consequences, and that exceptions are no longer as easy to handle as they once were. More significantly, users may be led to depend on a computer system that acts according to a patched-together theory of their work that is entirely unclear to them. If there is no well formed theory of use behind a computer system, then the operation of the system is little better than experimental; in such cases, dependance on a computer system seems highly irresponsible.

2.2.5: Systems

Before going any further in this discussion, I should say something about the important distinction between computer programs and computer systems. Much of the incomprehensibility on which concerned computer scientists end up focusing is that of computer programs and, to a smaller extent, of the processes of problem analysis, design, and programming itself, which culminate in the generation of program code. We are exercising extremely narrow vision in restricting our view of computer system incomprehensibility to the incomprehensibility of one part (albeit the most logical and well understood part) of computer systems - computer programs.

Butler Lampson, a senior research fellow of the Xerox Corporation, has stated his belief that "it is the source text that completely defines the [computer] system."[14] It is not surprising that his ideas about making computer systems more comprehensible involve schemes for things like building structure into program code (through greater attention to program hierarchy and interfaces between programs). Lampson claims that by making explicit all changes to code (for instance, by

---

[14]Butler Lampson, "Building Programs," MIT Laboratory for Computer Science Distinguished Lecturer Series, May 1, 1979.

keeping track of the editor in some well defined way), we will be able to explicitly capture "the nature of the changes," and thus figure out what is going on. Lampson's concerns about the dynamic nature of program evolution are valid ones. However, his persistence in keeping everything but computer programs themselves out of his treatment of incomprehensibility ignores the fact that a complete system is more than just computers and the other machinery associated with their operation.

Even assuming that there existed unambiguous, organized methods of making computer programs understandable (and this in itself presupposes that we know and agree upon what it means to understand a computer program), a computer system cannot be rendered comprehensible simply by "building" understandability into its software. For instance, I have already mentioned the importance of taking into account the context of use of a computer system in any discussion of the system.

> . . . most large systems actually in use today and on which people depend . . . simply are not logically deterministic systems in any useful sense of the term. It no longer makes sense to speak of such systems as having a state at a particular time, or of their programs as if these were concrete texts having an existence independent of the detailed circumstances in which they are used.[15]

T. D. Sterling, writing in a recent article on the social impacts of computing, expresses my own desire for a wide view of computer systems; he repeats Kling and Scacchi's definition of a computerized "package": "not only devices (*e.g.*, hardware, software, and systems protocols), but also a diverse set of skills, organizational units to supply and maintain computer based services and data, and sets of belief of what computing is good for and how it may be used efficaciously."[16]

---

[15]Joseph Weizenbaum, "Human Choice in the Interstices of the Megamachine," pp. 12-13. Lecture presented at the IFIPS Conference on "Human Choice and Computers," Vienna, Austria, in June, 1979.

[16]T. D. Sterling, "Consumer Difficulties with Computerized Transactions: An

Like these people, by referring to computer systems I mean to suggest something with many dimensions, some of which have found no place in purely technical discussions of computer incomprehensibility. The all-encompassing nature of computer systems is what makes it so difficult to define and to deal with the incomprehensibility that derives from them, but the frequent tendency to shy away from consideration of the societal aspects of computers has helped entrench the incomprehensibility of some systems.

Empirical Investigation," *Communications of the ACM*, 22 (May, 1979), 284.

Chapter 3: Sources of Incomprehensibility

> Evidently technological accomplishment has become a temptation that no person can reasonably be expected to resist. The fact that something is technically sweet is enough to warrant placing the world in jeopardy.[1]
>
> Langdon Winner

It is hard to realistically deny the pervasiveness of incomprehensible computer systems in modern society. Most of the computer scientists with whom I have spoken have ready examples of their pet incomprehensible systems (large operating systems are favorite choices). In discussing incomprehensibility, many of these people smile knowingly at me as they remark that it certainly isn't difficult to find instances of baffling computer systems. What I find particularly upsetting about these conversations is the fact that the comments are made from a professional viewpoint; that is, the systems to which these people refer are felt to be incomprehensible to computer specialists. What, then, is the position of all the non-technical people who make use of and are otherwise affected by large systems; how are these people to understand a computer system that the "experts" have called incomprehensible? If the seriousness of the problem of incomprehensibility is acknowledged by computer scientists, why is it that incomprehensible systems are so widely used today and that many systems currently being constructed are likely to emerge incomprehensible in some sense of the word?

In this chapter, I consider the question of how does incomprehensibility arise in a computer system. Like most questions that involve human interactions and needs (and those dealing with computer *systems* do), this one does not have a single, well defined answer. Rather, computer system incomprehensibility is a problem

---

[1] Langdon Winner, *Autonomous Technology* (Cambridge, MA: The MIT Press, 1977), p. 73.

that brings together a host of different issues. The "reasons" that we may come up with to explain the existence of incomprehensible systems depend on our point of view - that of a researcher, programmer, designer, businessman or other user, social critic, etc. For the purposes of this discussion, incomprehensibility is best considered from all these perspectives.

3.1: The Programming Process

Computer programs are among the most obvious components of computer systems, so we may begin an analysis of the sources of incomprehensibility at the level of computer programming. I have previously discussed some problems of communication that beset many large programming projects. Most software workers are aware that there are a variety of difficulties with virtually every stage of the construction of a computer system, and that these difficulties confound programmers working both individually and as part of a larger endeavor.

On a personal level, programmers are viewed by much of society as rather peculiar people. Computer scientists must cope with unproductive stereotypes that emphasize what might be called the "hacking mentality."

> . . . the hacker is "without definite purpose": he cannot set before himself a clearly defined long term goal and a plan for achieving it, for he has only technique, not knowledge. He has nothing he can analyze or synthesize; in short, he has nothing to form theories about. His skill is therefore aimless, even disembodied. It is simply not connected with anything other than the instrument on which it may be exercised.[2]

Programmers are sometimes perceived as having little purpose other than to spend time with computers; they may be seen as solitary individuals whose strong attachment to machines is proportional to their detachment from and lack of experience

---

[2]Joseph Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation* (New York: W. H. Freeman, 1976), p. 118.

with other people.

Professional isolationism is encouraged through the training of software workers, which single-mindedly emphasizes technical skills. Langdon Winner notes that engineers typically regard themselves as "merely problem solvers":

> "Tell us the problem," they demand. "We will find a solution. That's our job. But you may not presume to question the nature of our solution. You are not a member of a technical profession and, therefore, know nothing of relevance. If you insist on raising questions about the appropriateness of the means we devise, we can only conclude that you are antitechnology."[3]

The personal identification of programmers with their programs, which occurs in many existing set-ups, does not support productive communication between workers. In addition, workers in technological fields are subtly taught to accept segmentation in the work structure that surrounds them. Programmers are discouraged from asking questions or exploring issues beyond their assigned tasks; they are "deprived of all but the most narrow skills and of an understanding of how their work fits into the work process as a whole."[4] Coders may be forbidden to use any programming techniques but a few rigidly defined ones, so that their program code emerges as little more than "a standardized product made in in a standardized way by people who do the same limited tasks over and over without knowing how they fit into a larger undertaking."[5]

Even on a more organized level (*e.g.*, that of the management of a software project), the activity of programming is often viewed in an impersonal way. In his book, *The Psychology of Computer Programming* (which is, incidentally, one of the most widely read books about computer programmers), Gerald Weinberg comments

---

[3] Winner, p. 11.

[4] Philip Kraft, *Programmers and Managers: The Routinization of Computer Programming in the United States* (New York: Springer-Verlag, 1977), p. 83.

[5] *Ibid.*, p. 59.

on the widespread lack of attention devoted to the informal, interpersonal (as opposed to formal, organizational) structure of software projects and to the social (as opposed to physical) environment that supports the work of programming. The fact that Weinberg finds it necessary to stress the human element of programming is indicative of a general lack of understanding of and concern for programmers and the interactions between them, that is not uncommon in managerial circles. Philip Kraft, too, has much to say about the artificially maintained distance between programmers and managers, and its negative effects on the production of computer systems.[6]

In addition to psychological factors that play a significant role in the programming process, there are other, even more tangible pressures that affect the quality of computer programs, often by contributing to their inscrutability. The most obvious of these pressures are caused by short range economic constraints on all programming projects. These constraints discourage the writing of thorough documentation or of well commented, understandable code. It is a common folklore among people who are not terribly familiar with computer programming that good code is likely to be more expensive to run than code that consists of program "hacks,"[7] but this notion is not supported by practical experience. What is certainly true is that obscure code is in the long run costlier to maintain than well written code. Unfortunately, the non-monetary costs of using poorly coded programs (which are bound to be unreliable and incomprehensible) are not reckoned into the budgets of the projects that produce them.

Economic requirements translate into time limitations that tend to pressure

---

[6] Kraft, *op. cit.*

[7] A vernacular term used to denote quick, clever pieces of work that are generally not very well structured or documented.

software engineers into accepting inadequate solutions to problems that they are asked to solve. Weinberg voices the opinion of everyone who has ever done any serious programming when he states that "we must be realistic and acknowledge that probably no perfect program was ever written. Every really large and significant program has just one more bug."[8] Nevertheless, adequate time for careful program design, meticulous coding, extensive testing, and complete documentation - all of which are necessary if we are even to attempt to produce high quality, understandable code - is considered a luxury in most non-academic programming situations. In general, it is only the short term achievement of a system that runs which is tangibly rewarded. Because the long term, global view is not usually taken, important considerations may be overlooked. For instance, once a given computer system is declared operational and is put into use, many of the programmers who were involved in making that system run begin working on other systems; thus, the people who help design and implement a large system are not usually around when problems with the system arise.

Another factor that contributes to program incomprehensibility is the generally low level of professionalism that characterizes the activity of programming. We think of an activity as being professional when the members themselves determine the criteria for membership, choose who is to be a member and who is not, and have the power to remove someone from the profession if he does not adhere to certain collectively set technical and ethical standards. The Council of the Association for Computing Machinery has adopted a set of guidelines for professional conduct in information processing, but computer programming is not really governed by anything like a code of ethics; the fact is that there is no

---

[8]Gerald Weinberg, *The Psychology of Computer Programming* (New York: Van Nostrand Reinhold Company, 1971), p. 19.

professional standing for programmers (the ACM primarily serves academics and other computer science researchers;[9] that is, people who make up a small portion of all people who work with computers).

Current writings that deal with the training of programmers indicate that software workers are taught not to question the constraints that are imposed on their work, even though these constraints may be imposed by people who have had few dealings with computers; they are taught to stress corporate profitability and to identify this with technical rationality and efficiency, even though these criteria are not sufficient to determine when a given performance level of a program is met and can be depended upon; in short, they are taught everything but a sense of professional ethics. Furthermore, they are subjected to continually changing specifications from computer users; these, combined with the coming and going of people involved in a computer project as the design of the system progresses, ensure steady modification that fundamentally changes a computer system in unpredictable ways.

3.2: Complexity

Some advocates of increasing the use of computers would have us believe that computers are both necessary and beneficial to a society as complicated as our own. They argue, on the one hand, that "without such intellectual aids, our complex modern society may well fall apart under its own weight of complexity,"[10] and, on the other hand, that better (*i.e.*, computerized) means for handling and

---

[9]This can easily be verified by a casual survey of the affiliations of the authors of the articles and letters in ACM publications.

[10]Robert Fano, "The Computer Utility and the Community," *IEEE International Convention Record*, Part 12, 32.

disseminating information will result in an increase in intellectual freedom and in diversity of lifestyles. These arguments form the basis for much of present day computer use; yet, as I intend to show, they do not stand up to critical examination.

The value of the computer in dealing with really complex problems, and particularly with human difficulties, is highly questionable. In some cases, the complexity of computer applications derives not from the problem domain, but from the use of computers. It is actually the case that some computer systems generate complexity and render problems more incomprehensible and unmanageable than they were originally. For instance, Edwin Paxson attempts to justify computerized military defense systems by stating that "there is little question that serious systems analysis without the computer is impossible"; he goes on to undermine his own argument by unknowingly naming the computer itself as the source of the complexity with which it is associated: "As computing power has increased, so also has the complexity of the analyses it *fosters*."[11] [emphasis mine] Similarly, John Kemeny first praises Dartmouth College's computerized bookkeeping system for maintaining many thousands of accounts accurately, and then acknowledges that "It is also a by-product of the system that we have such a vast complexity of information . . . that human beings cannot cope with it without the aid of the computer."[12] In the light of what these people and others have said, the argument that computers are necessitated by the complexity of modern society is hardly tenable.

The alleged benefits of complex computerized information systems are even

---

[11]Edwin W. Paxson, "Computers and National Security," Chapter 3 in *Computers and the Problems of Society*, ed. by Harold Sackman and Harold Borko (New Jersey: AFIPS Press, 1972), p. 77.

[12]John G. Kemeny, *Man and the Computer* (New York: Charles Scribner's Sons, 1972), p. 104.

easier to dismiss than the preceding "argument from necessity." According to one view, computers, by virtue of their information processing capabilities, are part of the foundation of a new golden age. However, it is not difficult to see that existing computer systems do not provide any evidence of the coming of this prosperous future, and there is no compelling reason to believe that twice as many, or even ten times as many systems ever will.

> The record of how technology has actually been used in the modern world does not support [this] supposition. In principle, transportation and communication might have been expected to contribute to a broadening of human experience and greater tolerance for different customs and beliefs . . . . The ability to travel with ease on a global scale and communicate across great distances has not led to increased understanding or great compassion. There is no evidence to suggest that international tensions have been lowered as a consequence of increased information flow . . . .
>
> The problems of human interaction cannot be reduced to information flow, nor can the shortcomings of existing social arrangements be attributed to imperfections in our instruments of communication. Is there any reason to believe that a two-way terminal in the home would materially alter an individual's response to television broadcasts showing scenes of violence or human mercy? The mere fact of having access to information does not create the disposition to act appropriately . . . . It is purely wishful thinking to suppose that improved information flow will result in spontaneous efforts to resolve conflict and create more responsive social environments. The historical evidence points to further concentration of power."[13]

Mowshowitz drives home the point that although information is necessary for rational decision making, it is not sufficient for harmonious social interaction.

In discussing complexity, I do not mean to refer to the formal definitions of complexity utilized by computer science researchers, but rather to a more intuitive definition. A complex computer system is one that is composed of many parts that are interrelated in complicated ways, so that it may very well be impossible for us

---

[13]Abbe Mowshowitz, *The Conquest of Will: Information Processing in Human Affairs* (USA: Addison-Wesley Publishing Company, 1976), pp. 164-65.

to keep track of all the systems' components and the relationships between them. In reference to truly complex computer applications, I have already discussed the importance of a well understood theoretical foundation; a computer model (or any model, for that matter) is only as good as the theory behind it. The point here is that complexity is not a mandate for incomprehensibility: witness the MACSYMA system (mentioned in Chapter 2) and the DENDRAL system, which implements a theory of mass spectrometry. " . . . incomprehensibility is not a necessary property of even huge computer systems. The secret of their comprehensibility lies in that these systems are models of very robust theories."[14]

Most computer systems in use today "don't deal with complexity at all - nor are they designed to - they deal with sheer magnitude."[15] This is often the case with the huge information banks utilized by so many businesses. In these applications, computers have become necessary only in the sense that certain services could not be rendered in their present form without the computer. However, the present form of these functions has been dictated in large part by the availability of and early dependance on computers. This state of widespread computer use and dependance, which now appears inevitable and inescapable, is in reality man made. Apparent complexity may be deceiving and does not always warrant the use of computers, which may only conceal less technological, perhaps more perspicacious ways of solving certain problems.

I have mentioned complexity in my discussion of incomprehensible computer systems because it is one quality that is nearly always associated with computers

---

[14]Joseph Weizenbaum, "Human Choice in the Interstices of the Megamachine," p. 13. Lecture presented at the IFIPS Conference on "Human Choice and Computers," Vienna, Austria, in June, 1979.

[15]*Ibid.*, p. 7.

and that figures largely in their incomprehensibility. Shallow understanding of a complex problem may result in a system that is unnecessarily opaque. Unfortunately, our assumption that computer systems are too complex for the average person to understand has helped prevent the exposure of much existing system incomprehensibility. In dealing with computer systems, it is well to consider both the easily forgotten fact that enormously complicated tasks were successfully carried out before the advent of computers, and the easily obscured fact that the modern megamachine[16] has not heralded an age of human happiness.

3.3: Evolutionary Systems

One of the reasons that even computer scientists are worried about incomprehensible systems is the current reality of computer systems that have grown to a point where they are no longer under the control of the people who conceived and created them. A large system is worked on by so many people over such a long period of time that by the time it is completed, there is no person or small group of people who can be said to understand it.

One of the most disturbing facts about large computer systems is that there is no group of people who can be identified as the authors of the system; in other words, no one is ultimately responsible for the operation of the system. Virtually all large computer systems are far too difficult to comprehend for anyone to be able to modify them without risking unpredictable consequences; certainly this is always the case with systems of whose evolution we cannot keep track. Once they are put into operation, these systems are, for the most part, immune to change - they

---

[16] A term used by Lewis Mumford to describe massive organizations intended to carry out tasks whose magnitude places them beyond the capabilities of small groups of people.

can only grow.

Herbert Simon, a leading computer specialist, describes a computer program as a strategy of action whose states and inputs need not be envisioned in advance by the programmmer.[17] Such an attitude does not support the security derived from the commonly held view that computers do only what they are told to do. In a fearfully real sense, Simon is describing systems whose present form and mode of operation were neither planned not foreseen when the system was designed. These are systems that have *evolved* into what they are now, in ways that no one has kept track of or really understands.

> When a program grows in power by an evolution of partially understood patches and fixes, the programmer begins to lose track of internal details, loses his ability to predict what will happen, begins to hope instead of know, and watches the results as though the program were an individual whose range of behavior is uncertain.
>
> This is already true in some big programs . . . it will soon be much more acute . . . large heuristic programs will be developed and modified by several programmers, each testing them on different examples from different consoles and inserting advice independently. The program will grow in effectiveness, but no one of the programmers will understand it all. (Of course, this won't always be successful – the interactions might make it worse, and no one might be able to fix it again!) Now we see the real trouble with statements like "it only does what its programmer told it to do." There isn't any one programmer.[18]

Professor Minsky might well have said that there are no individual programmers at all. Gerald Weinberg has emphasized the team programming concept;[19] the survival of the team throughout all phases of a computer system, from design through

---

[17]Herbert A. Simon, "What Computers Mean for Man and Society," *Science*, 195 (March 18, 1977), 1187.

[18]Marvin Minsky, "Why Programming is a Good Medium for Expressing Poorly Understood and Sloppily Formulated Ideas," in *Design and Planning*, II, ed. by M. Krampen and P. Seeltz (New York: Hastings House, 1967), p. 121. Quoted in Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation*, p. 235.

[19]Weinberg, *op. cit.*

maintenance, encourages the belief that there is a group of people who continually maintain control over the system. However, the composition of a team changes so often that the team that maintains a system and must deal with problems that arise in the course of using it is fundamentally different from the team that made the initial design decisions. Thus, in addition to the fact that large computer systems adapt as they operate, the group of people that is supposedly in control of the system continually changes. It may be impossible to reliably predict the final results of a computer system for quite some time after it has been put into operation and after people have begun depending on it; it may also be impossible to reliably predict how well a programming team understands a system at a given time.

Much of the incomprehensibility that characterizes these evolutionary computer systems derives from the organization of the programming of the system - the highly segmented, hierarchical structure of a large computer project - and from the programmers themselves - the constantly changing nature of the group of people who create and then maintain a computer system.

> Most of the routine and tedium associated with the daily tasks of producing programs are left to an anonymous army of people who merely do what they are told, understanding little of what they do and less of why they are doing it. At least up until now, the computer has intensified, not reduced, the separation between those who think and those who do everything else.[20] [my emphasis]

Although such systems may not be smart enough for us to imagine depending on them a great deal, Marvin Minsky worries that "unfortunately, there are too many ways a dumb system with a huge data base can be useful."[21] Just as one

---

[20] Kraft, p. 29.

[21] Marvin Minsky, "Computer Science and the Representation of Knowledge," in *The Computer Age: A Twenty-Year View*, ed. by Michael L. Dertouzos and Joel Moses (Cambridge, MA: The MIT Press, in press). Quoted in Joseph Weizenbaum, "Once More: The Computer Revolution," in Michael L. Dertouzos and Joel Moses, ed., *op. cit.*

cannot talk usefully about a computer program without also talking about the technological system in which it is embedded, one cannot talk usefully about a computer program without considering the database on which it draws for information. It is often the case that one cannot distinguish very easily between the program and the database system. Modern databases are often dynamic, introducing yet another source of unpredictability into a computer system.

It is not difficult to find people who seriously worry whether we can maintain control of our computer systems - whether people can keep up with their machines. It is harder, however, to find someone who recognizes the absurdity of needing to ask such questions.

> But we must win our technological race with competing nations first and then do the best we can with the realignment problem. Remember, the readjustment problem is common to all technologically-advancing nations.[22]

What Thompson euphemistically calls the "realignment problem" is really what Joseph Weizenbaum describes as "the feeling of powerlessness so ubiquitous among individuals in our society . . . the widespread alienation of people from one another and from their work . . . the perception of ordinary people that they are living in the interstices of a gigantic system."[23] Computer systems usually replace older, less technological ways of doing things, and once a workplace is organized around computers, they quickly become indispensable to the functioning of that workplace. If no one is willing to assume responsibility for a system, why are we willing to trust that system in non-trivial areas of our lives?

---

[22]Howard Thompson, *Joint Man/Machine Decisions* (Cleveland, Ohio: Systems Procedures Assoc., 1966), p. 57.

[23]Weizenbaum, "Human Choice in the Interstices of the Megamachine," p. 1.

This empty page was substituted for a blank page in the original document.

Chapter 4: The Technological Society

> Of all the things inappropriate to the man-made environment of the modern age, none is so inappropriate as man himself. 'He must adapt himself,' Ellul comments, 'as though the world were new, to a universe for which he was not created.'[1]
>
> Langdon Winner

> Quietly and complacently, it was sinking into decadence, and progress had come to mean the progress of the Machine.[2]
>
> E. M. Forster

I have already noted that computer programs should not be considered independently of the circumstances in which they are used; similarly, computer systems and system incomprehensibility should be examined from a broad perspective that emphasizes the technological society in which we live. Daniel Bell writes that " . . . technology is not a 'reified thing' or some abstract 'logical imperative' but is embedded in a social support system, and it is the support system, not the technology, that determines its use."[3]

There are many aspects of society's attitude toward technology that can lead directly to incomprehensibility in computer systems. Our technological society operates in "performance mode," whereby issues not directly related to the economic value of a system are likely to be suppressed. Questions such as whether or not we need a particular system or whether a system is safe (in a broader sense than that exemplified by whether a system is harmless to its

---

[1]Langdon Winner, *Autonomous Technology* (Cambridge, MA: The MIT Press, 1977), p. 216. Quoting Jacques Ellul, *The Technological Society*, trans. by John Wilkinson (New York: Alfred A. Knopf, 1964), p. 325.

[2]E. M. Forster, "The Machine Stops," in E. M. Forster, *The Eternal Machine and Other Stories* (New York: Harcourt Brace Jovanovich, 1928), p. 285. Quoted in Abbe Mowshowitz, *The Conquest of Will: Information Processing in Human Affairs* (USA: Addison-Wesley Publishing Company, 1976), p. 313.

[3]Daniel Bell, "Hard Questions and Soft Minds: A Reply to Weizenbaum," Chapter 21 in *The Computer Age: A Twenty-Year View*, ed. by Michael L. Dertouzos and Joel Moses (Cambridge, MA: The MIT Press, in press).

operators, for example) may be subdued by questions like "will it work?"

There is widespread pressure against critics of computer systems, which encourages dependance on these systems. This often takes the form of peer pressure; counter critics cite the observable fact that "everyone else" seems to be using computers.

> And so we conclude that on-line decisions have to be made rapidly in some systems because of the advance in technology, and have to be made rapidly in business to meet the competition created by learning to make those vital, rapid decisions elsewhere.[4]

The above reasons - business competition and advances in technology - are typical but not extremely compelling justifications for the often uncritical expansion of large scale computer applications and for the discouragement of criticism.

4.1: Autonomous Technology

The organization of modern society is founded on what many people perceive as technological necessity - a widespread sense of technological inevitability that Langdon Winner has called "autonomous technology":

> Autonomous technology is the part of our being that has been transferred, transformed, and separated from living needs and creative intelligence. Any effort to reclaim this part of human life must at first seem impractical and even absurd.[5]

Several writers have distinguished between technology and the broader connotations of technique - a limitless way of organizing the world; an all-encompassing arrangement of which human society is but one segment. My intent throughout this chapter is to examine the presence of technique in today's society. Most of us

---

[4]Howard Thompson, *Joint Man/Machine Decisions* (Cleveland, Ohio: Systems Procedures Assoc., 1965), p. 40.

[5]Winner, p. 333.

are governed to a larger extent than we consciously acknowledge by a technological mentality, according to which technique assumes the characteristics of an irresistible force and not of just a tool. Our sense of inner directedness has been superseded by a sense of what we believe our highly organized environment "wants" us to do or will allow us to do.

The technological society is, to a large extent, a dangerously uncritical one. We have allowed our virtues to become technical ones; the spirit of the day is that of maximum productivity. Technique "clarifies, arranges, and rationalizes; it does in the realm of the abstract what the machine did in the domain of labor."[6] Technique specifies attitudes that are valid once and for all.

Winner exhorts us to consider "instances in which things have become senselessly or inappropriately efficient, speedy, rationalized, measured, or technically refined."[7] I have encountered numerous examples of a thoughtless acceptance of technological virtues; I will mention only a few of them. Howard Thompson (in *Joint Man/Machine Decisions*) implicitly assumes the desirability of complexity and competitiveness; on the basis of this assumption, he does not bother to justify the use of computer systems in his discussion of decision making.[8] The title of Sheridan and Ferrell's book, *Man-Machine Systems: Information, Control, and Decision Models of Human Performance*, would seem to indicate an emphasis on the human element of man-machine interactions. However, the authors inform us that their consideration of human performance is necessarily limited to that which their models can describe:

---

[6]Jacques Ellul, *The Technological Society* (New York: Vintage Books, 1964), p. 5.

[7]Winner, p. 230.

[8]Thompson, *op. cit.*

> People may show grace, imagination, creativity, or feeling even in narrowly constrained tasks; but these qualities are too fine for the nets we cast in modeling and experiment. We have to be content to describe and predict at a much more mundane level. Our frequent use of terms such as *operator* and *performance* instead of *person* or *behavior* is meant to emphasize the engineering context and the relatively narrow range of human experience which it encompasses.[9]

Sheridan and Ferrell certainly deserve credit for recognizing and bringing to our attention some of the distinctions between the human and the mechanical elements of the interactions that they consider. Still, when we are led to believe that we must be content with a narrow range of human experience because technique requires this limitation, then our lives are being viewed too much in the "engineering context." ". . . to dwell on these impressive statistics which tell us what people *do*, without attention to how they *feel* about what they do is to miss a profoundly important dimension of human experience - that is, the *meaning* that people attribute to their behavior."[10]

In one article, Harvey Wheeler attempts to explain why we should use computers to aid us in making decisions. He states that we do not expect perfection from either men or machines, but that all we want is a way to make decisions more systematically.[11] Again, I call attention to the unquestioned assumptions - in this case, that systematicity as such simply and obviously justifies itself and is always to be preferred over less formal criteria. Why not go beyond the question of whether or not we *can* systematize the way in which we make decisions and ask

---

[9]Thomas B. Sheridan and William R. Ferrell, *Man-Machine Systems: Information, Control, and Decision Models of Human Performance* (Cambridge, MA: The MIT press, 1974).

[10]Lillian Breslow Rubin, *Worlds of Pain/Life in the Working-Class Family* (New York: Basic Books, Inc., 1976), p. 135.

[11]Harvey Wheeler, "Artificial Reasoning Machines and Society," Chapter 13 in *Computers and the Problems of Society*, ed. by Harold Sackman and Harold Borko (new Jersey: AFIPS Press, 1972), p. 489.

whether or not we really want or ought to do so?

The use of computers is sometimes justified by the argument that at least computers are no worse than people (*e.g.*, automated decision making can hardly be worse than decision making by executives in a complex bureaucracy), and that the computer's rationality frequently makes them preferable to people in some applications.[12] Often, there is evidence of a pre-determined approval of the use of computer technology that may cause us to ignore many undesirable aspects of applications that computer systems efficiently cover up - aspects that cannot be effectively dealt with by a computer system. For example, the ELIZA computer program (mentioned in Chapter 2) was hailed by some people as the precursor of a psychotherapy machine that would deal with much of the neuroticism of modern society better than human therapists are able to do. People who made this judgment tended to lose sight of the nature of human psychological problems amidst their raptures over the computer. "ELIZA had less to do with showing how much a computer can do than with revealing how cognitively and emotionally empty some forms of human interaction can be . . . ."[13]

In our society, there is a strong temptation to submerge one's own autonomy in the megamachine; to give way to "a conscious and unconscious response to whatever situation arises. This response strongly and automatically repulses any alternative mode of action . . . it neutralizes alternatives by making them seem unnatural, impractical, or simply impossible."[14] In minimizing the role of human

---

[12]Several writers have argued that, although computer systems may not perform as well as superior human beings, the systems are better than most people. See Kenneth Colby, "Computer Psychotherapists," UCLA Department of Psychiatry, Algorithmic Laboratory of Higher Mental Functions Memo ALHMF-14.

[13]Theodore Roszak, "The Computer - A Little Lower Than the Angels," *The Nation*, 222 (May 1, 1976), 534. Review of *Computer Power and Human Reason: From Judgment to Calculation*, by Joseph Weizenbaum.

[14]Winner, p. 126.

beings in controlling their technical creations (be they computer systems or other technical systems), autonomous technology enables us to convince ourselves that the system progresses independently of our actions, and thereby to deny personal responsibility for our actions.

> It disturbs me to be told that technology "demands" an action the speaker favours, that "you cannot stop progress." It troubles me that we are so easily pressured by purveyors of technology into permitting so-called "progress" to alter our lives without attempting to control it - as if technology were an irrepressible force of nature to which we must meekly submit. If we reflected, we might discover that not everything hailed as progress contributes to happiness; that the new is not always better nor the old always outdated.[15]

Finally, in a society in which technology has become an autonomous force, the only ruling principle appears to be that the technological system must be expanded, at whatever cost. Human agents are permitted to make decisions only according to criteria related to maximizing technological efficiency; but this is not real choice.[16]

> To maximize energy, speed, or automation, without reference to the complex conditions that sustain organic life, have become ends in themselves . . . . Under the pretext of saving labor, the ultimate end of this technics is to displace life, or rather, to transfer the attributes of life to the machine and the mechanical collective, allowing only so much of the organism to remain as may be controlled and manipulated.[17]

Mumford uses the phrase "the new Megatechnics" to describe the modern system whose chief purpose is control over the physical world, and ultimately over man himself.

---

[15]Vice-Admiral H. G. Rickover, "A Humanistic Technology," in *Technology and Society*, ed. by Noel de Nevers (USA: Addison-Wesley Publishing Company, 1972), p. 23.

[16]Ellul, p 80.

[17]Lewis Mumford, "Authoritarian and Democratic Technics," in *Technology and Culture*, ed. by Melvin Kranzberg and William H. Davenport (New York: Schocken Books, 1972), p. 56.

4.2: The Computer Revolution

Computers are perhaps the most powerful playthings generated by modern technology. Unfortunately, computer systems are often used simply because they are there. We are too frequently guided by the technological maxim that any new technology that is possible must necessarily be utilized, whether or not it is needed or desired.

> If a machine can yield a given result, it must be used to capacity, and it is considered criminal and antisocial not to do so. Technical automatism may not be judged or questioned; immediate use must be found for the most recent, efficient, and technical process.[18]

Thus, Butler Lampson explains that we will see increasingly large computer programs in the future, *because* of the availability of more and better computer hardware,[19] and Benjamin M. Rosen, an electronics-securities analyst, says that Americans will indeed find a need for computers at home *because* technological advances are being made so rapidly.[20] One might ask why technological advances are being made so rapidly and discover that the rapidity of technological advancement is encouraged by the *a priori* perception of a societal need for the fruits of this advancement; thus, the minute size of this vicious circle is exposed. What is technically feasible is allowed to happen without regard to consequences.

We have already seen that economic incentives and social pressures combine to demand the use of computer systems. This situation has reached a point where any answer generated by a computer is often acceptable to us because of

---

[18] Ellul, p. 80.

[19] Butler Lampson, "Building Programs," MIT Laboratory for Computer Science Distinguished Lecturer Series, May 1, 1971.

[20] Mitchell Lynch, "A Computer Error: Trying to Use One in Your Home," *Wall Street Journal* (May 14, 1979), 33.

its automatic component. The worst charge of a technological society is that of the impedance of technical automatism. In a very real sense, we cannot afford to doubt our computers.

The so-called computer "revolution" consists largely of computer applications that are primarily guided by an implicit belief that computers should be used. This assumption is evident in many writings on the use of computers in psychotherapy, education, and military situations. Writers appear to start with the unjustified assumption that computers ought to be used to solve a given problem, and proceed to select for further consideration only those aspects of the problem that appear computable (refer to the following section for a discussion of how "the problem" is chosen in the first place). The problem itself almost appears to merit a status secondary to that of the computer.

> Data models are tools. They do not contain in themselves the "true" structure of information . . . . He [a user] has to learn how to use it. We generally presume that this learning is required only because of the complexity of the tool. Difficulties are initially perceived as a failure to fully understand the theory; there is an expectation that perseverance will lead to a marvelous insight into how the theory fits the problem. In fact, much of his "learning" is really a struggle to contrive some way of fitting his problem to the tool: changing the way he thinks about his information, experimenting with different ways of representing it, and perhaps even abandoning some parts of his intended application because the tool won't handle it. Much of this "learning" process is really a conditioning of his perceptions, so that he learns to accept as fact those assumptions needed to make the theory work, and to ignore or reject as trivial those cases where the theory fails.[21]

The question of whether a computer system is the best solution, or even an appropriate solution, to a problem, and the question of whether we should direct our efforts elsewhere instead of forging ahead with another computer system, are not

---

[21]William Kent, *Data and Reality* (New York: North-Holland Publishing Company, 1978), pp. 194-195.

always asked when they should be.

The usefulness of the computer is often assumed perhaps because of the awe with which our society generally regards science and technology. In our eagerness to find solutions to perpetual problems, we turn to what we believe we do best; in modern society, this is determined by the production of solid results. " . . . in the West at least, the test is not so much *what do you know?* or *how elegant is your interpretation of worldly phenomena?* but rather *what can you actually do?*"[22] Thus, we assume that a theory of any interpersonal activity can be expressed in the form of a computer program; that improved means will triumph over carelessly considered ends; and that what appears to the average person as the formal eloquence of the computer and the system behind it has the power to transmute errors into truths.

It is in the realm of social difficulties that the computer revolution is particularly inappropriate. One computer scientist has described a not-too-distant future in which we will have access to a general purpose computing language that can describe any system that can be imagined.[23] In this way, we are encouraged to believe that a more fully computerized world will eliminate existing social problems, " . . . to delude [ourselves] that gigantic instruments can take the place of no ideas at all."[24] What we are not encouraged to do is to come to grips with the human sources of these problems, to consider how subjective, interpersonal factors may be computed, or to examine criteria according to which social problems may be considered "solved."

---

[22]Winner, p. 25.

[23]Lampson, *op. cit.*

[24]Joseph Weizenbaum, "Human Choice in the Interstices of the Megamachine," p. 14. Lecture presented at the IFIPS Conference on "Human Choice and Computers," Vienna, Austria, in June, 1979.

Norbert Wiener points out some of the important ways in which social activities differ from the activities of computers:

> . . . learning machines must act according to some norm of good performance. In the case of game-playing machines, where the permissible moves are arbitrarily established in advance, and the object of the game is to win by a series of permissible rules according to a strict convention that determines winning or losing, this norm creates no problem. However, there are many activities that we should like to improve by learning processes in which the success of the activity is itself to be judged by a criterion involving human beings, and in which the problem of the reduction of this criterion to formal rules is far from easy.[25] [I would add that such reduction is usually not possible at all]
>
> To assign what purports to be precise values to such *essentially* vague quantities is neither useful nor honest, and any pretense of applying precise formulae to these loosely defined quantities is a sham and a waste of time.[26] [my emphasis]

Abbe Mowshowitz criticizes "automated common sense" - the substitution of formal processes for the intuitive decision making of an experienced manager. It is simply not always the case that all - or enough - of this intuitive knowledge can be made explicit; more often than not, valuable information is lost in the translation to the language of the computer.

4.3: Defining the Problem

A noticeable effect of the computer revolution is the frequent transformation of human difficulties into a form that is amenable to a computerized remedy. By omitting the step of convincing ourselves that a given problem really is technologically based, we are committing ourselves to forcing some problems into a

---

[25] Norbert Wiener, *God and Golem, Inc.* (Cambridge, MA: The MIT Press, 1964), pp. 76-77.

[26] *Ibid.*, p. 91.

mold - that of a computer system - in which they may not fit. The nature of such problems can easily be subdued by the method of approach and the techniques employed.

Many writers have remarked that the technological society is one in which our needs and desires are inevitably formulated as technological problems. It is the solution, *e.g.*, the computer, that defines the problem.

> If the technique in question is not exactly adapted to a proposed human end, and if an individual pretends that he is adapting the technique to the end, it is generally quickly evident that it is the end which is being modified, not the technique.[27]

Writing about a more specific domain, Philip Kraft comments on the attempt to formalize the activity of programming by increasing the use of pre-tested software routines: "In effect, the use of canned programs represents a joint decision by software sellers and software buyers to make the problems fit the solutions at hand."[28] Note Kraft's perception of the fact that the existing situation is the result of *decisions* that have been made by people.

The area in which the computer shows up most glaringly as a solution looking for a problem is that of social problems, in which there is most often no perceived need for computation (for example, the "problems" that characterize our educational or legal systems). Quick technological fixes for social problems have a way of affecting and changing the problems in unforeseen ways. Furthermore, just as the application of patches to computer programs compounds their

---

[27] Ellul, p. 141.

[28] Philip Kraft, *Programmers and Managers: The Routinization of Computer Programming in the United States* (New York: Springer-Verlag, 1977), p. 35. Note that this difficulty is common to all situations in which standardized - not personalized - solutions are applied to problems. I believe that Kraft is primarily concerned with canned databases, management information systems, etc., and not with relatively harmless (and genuinely important) things such as canned mathematical routines.

incomprehensibility, so the use of an entire computer system as a patch solution to a deep human difficulty can set in motion a dangerous propagation of incomprehensibility.

In considering the application of computers to social problems, we should first contemplate our perception of human difficulties as "problems." Joseph Weizenbaum has pointed out that problems such as a series of mathematical equations have permanent solutions, but that this is not the case with interpersonal difficulties. These problems involve conflicts of interest between people and cannot be understood solely in information processing terms; these problems cannot be understood without first understanding people. Human problems (for instance, system incomprehensibility) are not "solved" in the computational sense of the word; what would constitute a solution to a social problem? Rather, they are transformed into other problems that may be easier to live with than those they replace.[29]

We fall into a mode of problem solving when we realize, at some subconscious level, that in the realm of social difficulties, the subjective nature of problem definition renders problems not just complex, but extraordinarily difficult to deal with. Computer systems are comparatively easy to deal with. Thus, we concentrate our energy on improving military command and control systems instead of questioning the need for ever more complex and advanced forms of destruction; we extol the virtues of computerized psychotherapy without examining the reasons why increasingly large numbers of people seek psychotherapeutic help; and we try to introduce computers into schools on a massive scale without ever facing the reality of what is happening in American schools in other than information theoretical terms. We can convince ourselves that computers can solve our problems as

---

[29]Joseph Weizenbaum, "The Last Dream," *The Conference Board Magazine*, XIV, No. 7 (July, 1977), 41.

well as or better than people only by substituting mechanical gadgetry for human attention and by practicing interpersonal activities in inhumane ways. "The world becomes computerized when all human problems are reduced to technological problems."[30]

4.4: The Reduction of Human Experience

Perhaps the most regrettable effect of the technological structuring of present day society is the reduction of people to information processing organisms. Intelligence, once thought to be the exclusive domain of human beings, is now often defined operationally, so that we may speak of intelligent machines. At the 1977 International Joint Conferences on Artificial Intelligence, AI researcher Edward Fredkin stated that the achievement of a "thinking machine" requires a combination of only engineering and science.[31] Simon and Newell's General Problem Solver computer program is an attempt to implement their belief that the elementary processes underlying human thinking are analogous to the information processing of a computer.[32] Dartmouth president John Kemeny sees no good reason for not assuming the intelligence of computers, because they manifest intelligence in a scientifically testable (which is to say, extremely limited) sense. Kemeny equates apparent randomness (what we might call incomprehensibility) in a computer system with intuition and creativity in people, and implies that random computer behavior is as desirable as creative human acts.[33]

---

[30]Joseph Weizenbaum, "Apollo Agonistes," lecture presented at SUNY in Albany, New York, April 20, 1979.

[31]Israel Shenker, "Man and Machine Match Minds at M.I.T.," *New York Times*, August 27, 1977, p. 8, quoting Fredkin.

[32]Herbert A. Simon, "What Computers Mean for Man and Society," *Science*, 195 (March 18, 1977), 1186.

[33]John G. Kemeny, *Man and the Computer* (New York: Charles Scribner's

What I object to in such observations is their implicit reduction of people and of human experience; their identification of "the scientific conception of valid experience with the whole of existence."[34] Roszak notes that in equating people with machines, we can either raise machines up to our level or lower ourselves; he laments that we have done the latter - that we have reduced all human culture to the machine's limited capabilities.[35] Roszak is hardly alone in perceiving the gradual adaptation of human needs, desires, and thought processes according to the demands of technique; for instance, Winner writes of the shaping of human consciousness within narrow technical channels.[36]

In technological terms, an individual's social worth is proportional to his "productive capacity in a competitive labor market."[37] The production norms dictated by the goal of profitability often conflict with spontaneity and personal creativity. Individual participation in the technological society is tolerated only according to the degree of an individual's subordination to the search for efficiency; only that which is controllable is allowed to remain in man. Kemeny suggests that we may have to accommodate the computer systems that will play an increasingly major role in our lives;[38] whatever cannot be adapted will be eliminated.

I have already mentioned that life in modern society is formulated as a suc-

---

Sons, 1972), pp. 11, 18.

[34]Abbe Mowshowitz, *The Conquest of Will: Information Processing in Human Affairs* (USA: Addison-Wesley Publishing Company, 1976), p. 276.

[35]Roszak, p. 533.

[36]Winner, p. 127.

[37]Mowshowitz, p. 250. In reference to the emphasis on the economic value of people, consider the following: "But where human lives are at stake, and particularly when these people have paid for their transportation, a considerably higher degree of safety is required." Thompson, p. 97.

[38]Kemeny, *op.cit.*

cession of problems to be solved. Humans are viewed as problem solvers only, and are encouraged to resist sloppy human reason and intuition in favor of the artificial reasoning of machines. Mazlish urges us to see man's nature as being continuous with his tools and machines.[39] People become interchangeable as they grow increasingly alienated from their work and from other people. Technological reductionism can have nothing but an erosive effect on the self image of people.

It is not only the human being, but the entire human experience which is viewed from a reductionist viewpoint. Ethical and moral traditions that are not cost effective become obsolete; beauty becomes that which is well adapted to use. "The virtues of slow information processing and labor done at a leisurely pace have long since been sacrificed to the norms of work appropriate to the electronic exemplar. The idea that a task is something to be pondered or even savored is entirely foreign to this mode of activity."[40]

In order to accept a perception of life limited by technique, "humanity . . . has or will soon have transferred all its attention to one aspect of its being - it has sacrificed emotion for rational thought."[41] Thus, Herbert Simon identifies our most challenging problem, "the scientific problem of our age - how to understand ourselves more deeply."[42] In identifying interpersonal understanding as a *scientific* problem, Simon is severely limiting the role of human beings in the most human endeavor of all - understanding ourselves. As Ellul wrote, "men do not need to

---

[39]Bruce Mazlish, "The Fourth Discontinuity," in J. Mack Adams and Douglas H. Haden, *Social Effects of Computer Use and Misuse* (New York: John Wiley & Sons, 1976).

[40]Winner, p. 205.

[41]C. C. Gottlieb and A. Borodin, *Social Issues in Computing* (New York: Academic Press, 1973), p. 265.

[42]Shenker, quoting Simon.

understand each other in order to carry out the most important endeavors of our times."[43]

[43]Ellul, p. 132.

Chapter 5: Results of Incomprehensible Systems

> . . . technique, as a result of the perfection of means which it has placed at the disposal of modern man, has effectively suppressed the respite of time indispensable to the rhythm of life; between desire and the satisfaction of desire there is no longer the duration which is necessary for real choice and examination. There is no longer respite for reflecting or choosing or adapting oneself, or for acting or wishing or pulling oneself together. The rule of life is: No sooner said than done. Life has become a racecourse composed of instantaneous variations of the universe, a succession of objective events which drag us along and lead us astray without anywhere affording us the possibility of standing apart, taking stock, and ceasing to act.[1]
>
> Jacques Ellul

The use of computer systems, many of which are incomprehensible to the people who work with them, is firmly established in today's society, so it is not surprising that these systems have had significant impacts on our society of users. I have already examined a variety of factors that help explain why, in many cases, these impacts are undesirable and unanticipated. For example, consider the following points: first, our motivations for using computer systems are often not related to the nature of the problem at hand, and second, our perception of problems is frequently distorted by a pre-determined bias in favor of technological solutions.

In the previous chapter, I discussed some characteristics of a technological society; in this chapter, I will narrow that discussion to a consideration of some of the effects of the widespread presence of large computer systems. Some of these effects have already been explored in my discussion of the sources of incomprehensible systems; this is appropriate because technological systems often nurture attitudes and create conditions which support their continued use and expansion. Thus, certain issues I have previously examined - for instance,

---

[1] Jacques Ellul, *The Technological Society* (New York: Vintage Books, 1964), pp. 329-330.

autonomous technology, pro-technology biases, and the maintenance of what is often only an illusion of complexity in computer systems - are factors which both lead to the generation and use of computer systems, and result from our continued utilization of and dependance on these systems. In a society dominated by a myriad of technologies, computers occupy a uniquely awe-inspiring position. This chapter is an examination of some difficult questions raised by our growing use of computer systems which we do not understand; among them, questions about dependance on automatically generated output and responsibility for decisions made with the aid of computers.

5.1: Rationality

Scientific explanations derive both their power and their limitedness from the method of abstraction and simplification by which science proceeds. The scientific method is tremendously useful, but in limited ways; only information which is, in Kemeny's words, "scientifically testable,"[2] can be utilized in the construction of scientific experiments, models, and theories. Such information constitutes only one small aspect - the scientifically quantifiable aspect - of the world. Real life situations are characterized by an extreme richness of knowledge, much of which is "unscientific" and hence essentially unsuited for technical manipulation. Just as the reach for technological omnipotence continues to require the reduction of human beings to that which technology can explain and control, so the manifestation of power that we attribute to computer technology has necessitated the reduction of problems to those with which a computer can deal.

One key to the seemingly universal applicability of computer systems as

---

[2]John Kemeny, *Man and the Computer* (New York: Charles Scribner's Sons, 1972), p. 11.

solutions to problems is that before the application of computers, the problem domain is limited by being "rationalized." Problems must be made sufficiently explicit for interpretation by a computer system; to computerize often means first to rationalize. In many cases, it is this initial organizational effort, and not any computer system at all, that serves as *the solution* to the problem. Moreover, if the large expenditures of time, effort, and money needed to set up a large computer system result in a colossal failure, this failure is usually not linked to the questionable appropriateness of using computers in the first place; instead, a new problem is formed - that of the insufficiency of the logical composition of the problem area (this new "problem," in turn, may be deemed suitable for computerization). For instance, in writing of existing failures in the use of computers in conjunction with medical practice, Abbe Mowshowitz states that "the promise is enormous, but much depends on rationalizing the organization of health-care services."[3] The successful computerization of health care services depends more directly on the human effort of organizing the field than on the secondary step of bringing in computers.

The dangers of excessive rationalization derive from the lack of consideration for values which do not seem quantifiable, and the consequent loss of information in a purely rational planning or decision making process. The limiting rationality that computer systems demand encourages us to disregard the most difficult - that is, non-computable - aspects of a problem. Weizenbaum emphasizes the fact that computers process only information, not meanings.[4] Technique requires that certain aspects of problems be ignored, and even more significantly, determines which

---

[3]Mowshowitz, *The Conquest of Will: Information Processing in Human Affairs* (USA: Addison-Wesley Publishing Company, 1976), p. 123.

[4]Weizenbaum, "Once More: The Computer Revolution," in *The Computer Age: A Twenty-Year View*, ed. by Michael L. Dertouzos and Joel Moses (Cambridge, MA: The MIT Press, in press).

problems are to be seriously considered. " . . . we have permitted technological metaphors to so thoroughly invade our thought processes that we have finally abdicated to technology the very duty to formulate questions."[5]

5.2: Believing Computers

Our inability to comprehend many existing computer systems means that we must rely on the correctness of the systems; we have allowed such systems to become indispensable to us.

> We can't count on making such complex computations manually when we know that an enemy will use a computer. So we must rely on a computer for speed, which makes our decisions totally dependant upon both the availability and the accuracy of the computer when rapid decisions have to be made.[6]

The obvious risk is that of the increased impact of system errors as we increase our dependance on computer systems.

Today, the population in general does not understand technological forces, but is kept submissive and content with the wide range of services offered. Different groups, such as business managers, must depend on the output of computerized information systems, although they do not have the time to supervise the collection of data or to satisfy themselves as to the reliability of their computer systems (nor are they encouraged to do so).

> . . . how utterly dependant we have become on our electronic super-tools, how essential we have permitted them to *become*, not that they were *needed* in the first place.[7]

---

[5]Weizenbaum, "On the Impact of the Computer on Society," *Science*, 176 (May 12, 1972), p. 622.

[6]Howard Thompson, *Joint Man/Machine Decisions* (Cleveland, Ohio: Systems Procedures Assoc., 1965), p. 40.

[7]Weizenbaum, "Human Choice in the Interstices of the Megamachine." Lecture presented at the IFIPS Conference on "Human Choice and Computers," Vienna, Austria, in June, 1979.

The issue of dependance takes on a special meaning in relation to incomprehensibility. We have already seen that large computer systems are too complicated for anyone to directly monitor their operation. Thus, our day-to-day contact with computers must proceed largely on the basis of our belief in their correctness. When the computer systems we use are not understandable to us, this belief reduces to faith in a technology that we have been taught is too complicated for anyone but a specialist to understand. We abdicate the responsibility for our decision making to a technology that frequently is not comprehensible, while at the same time attempting to maintain a feeling of control. Thus, we assure ourselves that "these versatile machines have become the galley slaves of capitalism."[8] The real situation, however, is that our sense of control is largely illusory, and that we have been and continue to be largely unjustified in transferring responsibility to computer systems. This was made abundantly clear during the Vietnam era, when the Chairman of the Joint Chiefs of Staff regretted that "it is unfortunate that we have become slaves to these damned computers."[9]

Many people are now making decisions to some extent on the basis of potentially unreliable computer generated output, and some of these decisions can have vital, nonreversible impacts. In some cases, " . . . computers can provide not only the information on which decisions are made but can themselves make decisions."[10] At the very least, we must ask what kinds of decisions, if any, computers ought to make; we must decide whether the increased risk of error is worth the alleged gain in precision and rationality, if we depend more and more heavily on

---

[8]"The Computer Society," *Time*, Vol. 11, No. 8 (February 20, 1978), p. 50.

[9]*Ibid.*, p. 46, quoting Admiral Thomas Moorer.

[10]Herbert Simon, What Computers Mean for Man and Society," *Science*, 195 (March 18, 1977), p. 1187.

computers; and we must worry that the goals of a computer system on which we rely may not be our own goals:

> A goal-seeking mechanism will not necessarily seek *our* goals unless we design it for that purpose, and in that designing we must foresee all steps of the process for which it is designed, instead of exercising a tentative foresight which goes up to a certain point, and can be continued from that on as new difficulties arise. The penalties for errors of foresight, great as they are now, will be enormously increased as automization comes into its full use.[11]

Langdon Winner warns of "the distinct possibility of going adrift in a vast sea of unintended consequences."[12]

5.3: Technique and Morality[13]

It is to a large extent the common perception of computers, which may not have much relation to actual computer systems, that has determined the degree of our reliance on computer systems and the extent to which we have transferred human responsibility to automatic systems. The issue of responsibility can refer to different things in relation to computer systems; for instance, the responsibility of people for the propagation of new systems. In considering the concept of autonomous technology, we saw that often new systems are constructed without explicit or conscious human approval. Not only do we have trouble comprehending the systems we use, but it is often difficult for us to understand how and why they are created.

---

[11]Norbert Wiener, *God and Golem, Inc.* (Cambridge, MA: The MIT Press, 1977), p. 63.

[12]Langdon Winner, *Autonomous Technology* (Cambridge, MA: The MIT Press, 1977), p. 89.

[13]"Technique never observes the distinction between moral and immoral use. It tends, on the contrary, to create a completely independent technical morality." Ellul, p. 97.

> Moira is at work here - a fate that employs the free action of men to bring about ends that carry an aroma of necessity.[14]

The modern emphasis on a scientific analysis of behavior undermines personal autonomy by placing control of human actions in the environment. Responsibility for the unbounded increase in our use of computer systems seems to fall only on a technology that is, or so it is claimed, value free.

Technological elitism is at the root of much of the widespread avoidance of responsibility for computer systems. We are taught that only the experts can know what is best for us; in Kenneth Laudon's words, there has been "a denigration of faith in the wisdom of ordinary citizens."[15] Increased centralization, which frequently results from the introduction of a computer system, formalizes and rigidifies the prominence of those people who can claim to understand computers. In *Social Issues in Computing*, Gottlieb and Borodin comment on the political power of technocrats: because politicians themselves usually have no technical expertise, it is the technologists who define the alternatives for all of us.[16]

The people who appear to be most directly accountable for complex computer systems - computer scientists and the researchers who determine the state of the art - do not always manifest attitudes that are as cautious or as humble as their positions seem to dictate. Joshua Lederberg, for example, has said that there is no difference between the things computer should not do and the things people should not do; the only important thing is to be sure the machines do not get out of

---

[14]Winner, p. 71.

[15]Kenneth C. Laudon, review of *The Conquest of Will: Information Processing in Human Affairs*, by Abbe Mowshowitz, in *Science*, 193 (September, 1976), p. 1111.

[16]C. C. Gottlieb and A. Borodin, *Social Issues in Computing* (New York: Academic Press, 1973), p. 223.

control.[17] It is not clear whose control Lederberg believes computers are presently subject to; many other people believe that computers have already grown beyond the limits of our control. Still others believe that some programs themselves are already exercising control: Herbert Simon says of automatic process control systems that "their programs retain control over the ongoing process."[18] When questioned about how far artificial intelligence systems could go, Simon, with apparent disregard for issues of responsibility, incomprehensibility, reliability, appropriateness of use, etc., replied only that "we'll know that when we're done."[19]

Since, as we have seen, a large system has no identifiable group of authors, there is usually no one who feels directly responsible for the output of the systems and for decisions which make use of that output. Currently, accountability for the reliability of computer systems is so vaguely defined and so well "distributed" that it is fundamentally nonexistent. This is so despite the fact that many people are concerned about our tendency to allow computer systems to become ultimate authorities which require little justification.

> If the activities carried out by computers cannot be readily monitored and guided by people, and if human processing of information cannot be easily intermixed with computer processing, computers tend to become unchallengeable authorities . . .[20]

Recall the discussion of theory of behavior in Chapter 2, where I noted that comprehensible systems are likely to be founded on well understood theoretical

---

[17]Lee Dembart, "Experts Argue Whether Computers Could Reason, and If They Should," *New York Times*, May 8, 1977, p. 34.

[18]Herbert Simon, "What Computers Mean for Man and Society," *Science*, 195 (March 18, 1977), p. 1187.

[19]*Ibid.*

[20]Robert Fano, "On the Social Role of Computer Communications," *Proceedings of the IEEE*, 60, No. 11 (November, 1972), p. 1251.

bases, so that they serve as models of a theory and not as unquestionable authorities. In the case of incomprehensible systems, we have noted that there is frequently no well based theory of use; the system itself is the theory of its use. Thus, whoever doubts the system finds himself in conflict not with a theory but with an enormous, incomprehensible programming patchwork. Nevertheless, some people have gone to the extreme of advocating that computers be held responsible for themselves; Howard Thompson believes in letting the machine be responsible for its share of the decision making load in joint man/machine decisions.[21] The question I must ask is what meaning could machine responsibility possibly have in a human world?

> The habit of speech, and it surely reflects a habit of thought, that makes instruments responsible for events, leads directly to speaking and thinking of science and technology as autonomous forces and to the idea of technological inevitability. It leads finally to the proposition that man is, after all, impotent to struggle with powerful impersonal agencies of his own making over which he has lost control, and that he is therefore justified in abdicating responsibility for the consequences of his acts.[22]

---

[21] Thompson, pp. 27-28.

[22] Joseph Weizenbaum, "Controversies and Responsibilities," *Datamation* (November 15, 1979), p. 173.

This empty page was substituted for a blank page in the original document.

Chapter 6: What to Do About Incomprehensible Systems

> We must return to the human center. We must challenge this authoritarian system that has given to an underdimensioned ideology and technology the authority that belongs to the human personality. I repeat: life cannot be delegated.[1]
>
> Lewis Mumford

I would like to end my discussion of incomprehensible computer systems on a relatively positive note; in this final chapter, I consider suggested means of dealing with the existence of incomprehensible systems. The chapter begins with a discussion of program verification techniques that is more technical than that encountered in the rest of this thesis. Even in this area of formal study, many of the difficult problems are not technical ones and cannot be solved solely by studying computer programs: for example, the problem of how to specify what a program is *supposed* to do. This chapter, like previous ones, expands from a program-oriented viewpoint (which in this instance verification studies exemplify) to one concerned primarily with systems (both technical and social).

It should be noted that the following discussion pertains even to comprehensible computer systems; in fact, to all modern technological systems. Examination of well understood systems will generally reveal that many of the programming techniques described below were used, perhaps in modified forms, in the construction of the systems. However, evidence of the ethical "techniques" which are discussed in the latter part of this chapter (criticism of technology, acceptance of individual responsibility, and humanization of technological systems) is harder to find, even in systems that we might not label technically incomprehensible. Dealings with systems that are incomprehensible do raise unique ethical problems,

---

[1] Lewis Mumford, "Authoritarian and Democratic Technics," in *Technology and Culture*, ed. by Melvin Kranzberg and William H. Davenport (New York: Schocken Books, 1972), p. 58

because it is in these interactions that the necessity of trusting computers is implicit, and our vulnerability is greatest.

We already know that incomprehensibility in computer systems can be manifested in many ways. A social form of technological incomprehensibility is reflected by the pervasive sense that most people have of not occupying a meaningful position in modern technological society. For the most part, we do not understand how our technological systems operate - we do not know how to criticize technology, how to judge the extent to which we should depend on it, etc. It is especially because of these social effects of incomprehensibility that we feel no control over the use and expansion of computer technology and are unwilling to assume responsibility for it; it is these issues that are addressed in the second half of this chapter.

6.1: Program Verification

Verification has proved to be a difficult term to satisfactorily define, largely because of the human factors involved, first in the creation of the programs which are to be verified (programs carry with them their programmers' intentions, which are often unclear), and then in the interpretation of the proof of correctness (we want to be able to 'trust' programs, but we each have different criteria for believing a correctness proof). One suggested definition is the following: program verification is that branch of computer science whose goal is to establish "whether a [given] program performs its intended task."[2] What remain unanswered are the questions of what does it mean to talk about the intentions of a programming task,

---

[2]Barbara H. Liskov and Valdis Berzins, "An Appraisal of Program Specifications," MIT Laboratory for Computer Science CSG Memo 141 (July, 1976), p. 2.

and what does the assurance offered by a verification proof buy us?

The need for a discipline like program verification springs from two conditions that have existed for some time: (1) professional dislike for flaws in finished products (in this case, computer programs), and (2) increased dependance on programs that can have extremely destructive effects. On the one hand, there is a desire for certainty that has led us to mathematical formalism for a certification of program correctness. On the other hand (and more importantly), now that we have created computer systems that affect vital areas of our lives, we are beginning to wonder how we can depend upon the information that we get from computers. Because "even minor errors [let alone grossly misconceived "designs"] can have serious consequences and be costly to fix,"[3] the role of verification in increasing our trust in computer systems is both methodologically and ethically important. In terms of both physical and social costs, we cannot afford to trust unreliable computer software.

It is important that the reader recognize the inherent limitations of the verification approach to computer systems, before launching into the following discussion. Verification studies are directed toward computer programs. Verification researchers are committed to eliminating the relatively low level, technical form of incomprehensibility that is characterized by program errors. Recall from the first pages of this thesis that it is the incomprehensibility of systems with which I am primarily concerned; this is a much more subtle difficulty than that addressed by work in the field of program verification (in fact, as I have tried to explain, it is not a technical issue at all). Although there is a significant role that verification proofs

---

[3]Susan L. Gerhart and Lawrence Yelowitz, "Observations of Fallibility in Applications of Modern Programming Methodologies," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 3 (September, 1976), p. 195.

can fulfill, extrapolation from a proof of correctness to a meaningful statement about the comprehensibility of a system (and hence about the appropriateness of trusting that system) is, at best, unjustified.

6.1.1: Automatic Verification

The most widespread basis for verifying programs is the method of intermediate assertions. Tags are placed at key points throughout a program (*e.g.*, loops), indicating the state that the program is supposed to be in whenever it passes each point at execution time. The central idea is that all loops in a program must be "broken"; *i.e.*, it must not be possible to do a loop iteration without going through a tag. The intermediate assertions may relate values of program variables at intermediate points to initial values or to ultimate values. The program specifications for the initial state form the first assertion, and the desired output conditions form the final assertion. A proof of program correctness is divided into a number of smaller proofs that a program coming from assertion *n* will always satisfy the conditions of assertion *n+1*. The combination of these intermediate proofs establishes the partial correctness of the entire program.

Automatic methods of program verification center around the mechanization of the above approach (which might be called the informal assertion approach) through the use of automatic theorem-proving programs.

> The inductive assertion method reduces program correctness to a finite set of finite *paths*. A program *path* starts with an initial assertion, continues with executable code, and terminates with a final assertion. For each program path there is a logical formula called a *verification condition*.[4]

The first quest is for a system that would automatically generate the verification

---

[4]Ben Wegbreit, "Constructive Methods in Program Verification," Xerox Palo Alto Research Center (December, 1976), p. 6.

conditions. Once that is done, the proof of the correctness of a program reduces to a proof of a theorem in the first-order predicate calculus (higher-order systems are also being examined, but less effort is being devoted to them). The theorem establishes the partial correctness of a program, so a separate termination proof is required (in most discussions of program verification, it is deemed easier to divide a total correctness proof into two proofs, one of partial correctness - or correctness assuming termination - and one of termination). According to this view, a verification system consists of a verification condition generator and a theorem prover. In practice, the step of proving that the verification conditions are true has constituted the bulk of work in this area. The preliminary step of finding appropriate intermediate assertions is still too little understood to be automated.

There are a variety of problems, both technical and social, facing researchers in the field of automatic program verification. Some of the technical difficulties may only require further study to be overcome, but some of the constraints imposed by less formal problems may represent inherent obstacles to the success of verification proofs in increasing our trust in computer programs. Significant difficulties that are currently being addressed include the following:

- It is the hope of verification researchers that the verification of a computation is much easier than the original computation. However, time constraints place a heavy burden on any verification system. Although a given system may be able to verify a large class of programs, "we are more interested in what the theorem-prover can do in 'reasonable' time."[5] Thus, even a complete verification system would not necessarily satisfy the practical time limits that it would have to be subject to. Contemporary systems tend to flounder on complex programs which are

[5]Bernard Elspas, *et. al.*, "An Assessment of Techniques for Proving Programs Correct," *Computing Surveys*, Vol. 4, No. 2 (June, 1972), p. 127.

founded on deep theorems, because the search space is increased to a point where a proof cannot be generated in a tolerable length of time (and, even when it is generated, it is prohibitively long and complicated). "We simply do not yet see how to prove programs are correct in any reasonably short manner at the present time."[6]

• Until now, verification proofs have dealt with relatively simple cases (highly restricted programming languages and ideal machines), and are not yet up to the level of complicated programs, where they would be useful. Some relevant technical issues that are acknowledged as problems but have not been satisfactorily solved yet are indeterminacy, parallelism, exception errors (*e.g.*, overflow and underflow), and side-effects of a given programming language.

In addition, there is the difficulty of run-time errors and of specifying "the behavior of a program when an error is detected during execution."[7] Researchers in program reliability attempt to anticipate a range of possible execution errors ahead of execution time, so that error-handling measures can be included in the original design of the program. According to the philosophy underlying reliability work, errors are not necessarily eliminated, but they are anticipated and dealt with in understandable, acceptable, reliable ways.

• Some computer scientists are concerned about the believability of proofs, a criterion that would not be well satisfied by a system that receives a program whose correctness it is to establish as input and outputs a huge proof that is even less comprehensible than was the original program. A verification system's

---

[6]James Joyce, "Human Factors in Software Engineering," in *The First West Coast Computer Faire: Conference Proceedings*, ed. by John C. Warren, Jr. (Palo Alto, CA: Computer Faire, 1977), p. 61.

[7]Liskov and Berzins, p. 18.

response of "QED" is not very meaningful if programmers have no reason to trust the system. One primary function of a proof of correctness is "to dramatically increase one's confidence in the correct functioning of a particular piece of software,"[8] but before this can be accomplished, confidence in the proof process itself must be established. At worst, a complicated verification system could constitute an additional layer of incomprehensibility between a computer program and its users.

DeMillo, *et. al.*, convincingly argue that mathematical proofs come to be believed because of the existence of a social network in which proofs are widely read, refereed, published, reviewed, discussed, and finally, internalized, paraphrased, and used.[9] Although the motivations behind mathematical proofs and program verification proofs are different, the concept of believability is related similarly to both. Even a "correct" proof will not be used if it is not believed (belief here is defined in the social sense that DeMillo, *et. al.*, discuss), and DeMillo, *et. al.* do not believe that a social process in computer science analogous to that in mathematics is yet well developed. They also remind us that "the decision to consider a 'proof' in detail is often influenced by some slightly irrational concern - 'how does the problem *feel*?'"[10] and urge us to strive for the simplicity that characterizes the most important mathematical theorems and proofs. Finally, since any proof, no matter how formal, can be interpreted in different ways by different people, we should be aware that, in a practical sense, "'a derivation of a theorem or a

---

[8]Richard A. DeMillo, Richard S. Lipton, and Alan J. Perlis, "Social Processes and Proofs of Theorems and Programs," Yale University of Computer Science Research Report #82 (1976), p. 1.

[9]*Ibid.*, pp. 6-9.

[10]*Ibid.*, p. 10.

verification of a proof [can have] only probabilistic validity.'"[11] The classical view of judging things in a strictly dualistic fashion is not applicable to verification proofs of computer programs. We must be wise enough to recognize the margin of error in any assessment of program correctness, and to require "another view of 'reliable design' . . . that more fully exploits the social mechanisms"[12] to complement the view taken by verification studies.

• Computer systems are dynamic entities that may be vaguely specified. The foundations of modern "software systems are large programs with specifications and related documentation much larger than their code. More importantly, when specifying a system it is often impossible to state precisely what is to be done. Typically some claims are made about what must happen and others describe desirable but less crucial behavior."[13] In practice, specifications change and grow as a programming project progresses, reflecting a restructuring of the original purpose of the system and the original perception of the problem domain. The evolution of a system to meet the new criteria is usually not well controlled. "The incompleteness and imprecision of the specifications for systems makes rigorous verification difficult and the impermanence of the specifications reduces the rewards of producing such a verification."[14]

• The interface between an automatic verification system and a programmer must remain informal. What cannot be completely formalized in this interface is the purpose of the program, an informal, often unstated criterion. ". . . it becomes

---

[11] Gerhart and Yelowitz, p. 205.

[12] *Ibid.*, p. 197.

[13] Charles Rich, Howard E. Shrobe, and Richard C. Waters, "Computer Aided Evolutionary Design for Software Engineering," M.I.T. Artificial Intelligence Laboratory A. I. Memo 506 (January, 1979), p. 3.

[14] *Ibid.*

possible to formally prove consistency of programs with . . . formal specifications. However, the complimentary step of verifying that a program specification implements the underlying concept must necessarily remain informal."[15] We are faced with the problem of whether or not our specifications are strong enough to express our intentions. The most responsible attitude we can take is that "we can never be sure that the specifications are correct."[16] With such an attitude, we cannot be certain that a program that has been verified to be correct (assuming we achieve such results some day) will do what we want it to do, unless we are sure that our intentions have been accurately and completely codified in the program specifications.

> In verifying a program, the system assures us that the program satisfies the specifications we have provided. It cannot determine, however, whether those specifications accurately reflect the intentions of the programmer. The intentions, after all, exist only in the mind of the programmer, and are inaccessible to a program verification system. If he has made an error in expressing them, the system has no way of detecting the discrepancy.[17]

Some of the most promising current research in verification deals with ways in which a system could detect the kind of discrepancy mentioned above.[18] However, at least for the time being and particularly in the case of programs with vital consequences, an awareness of this 'discrepancy' should play a crucial role in our

---

[15]Zohar Manna and Richard Waldinger, "An Appraisal of Program Specifications," Stanford AI Lab memo AIM-298 (August, 1977), p. 24.

[16]*Ibid.*

[17]Gerhart and Yelowitz, p. 205.

[18]For example, the notion of a "programmer's apprentice," which is "a computer aided design tool which can help a programmer deal with program evolution from the initial design phase right through the continuing maintenance phase." Here, the effort is to provide "support during the process of developing code good enough to warrant the effort of certification." See Rich, Shrobe, and Waters, "Computer Aided Evolutionary Design for Software Engineering" (quotations are from pp. 1-2).

decision to depend on a particular computer program.

6.1.2: Informal Verification

What is urged throughout the field of program verification is the realization that "formalism should supplement, definitely not replace, common sense and programming experience."[19] Traditional verifying methods, which may all reduce to good programming standards, should be retained, along with a healthy skepticism towards formal proofs of correctness. Intuition or practical judgment can detect many programming errors. One of the most elementary insights that comes from a study of verification is that programming is a human activity and that there are parts of the programming process that are best handled with some measure of informality.

In the realm of informal, manual program verification, the most fundamental consideration is a statement of "what should be proved in order to guarantee that program is correct . . . ."[20] Since program specifications are generally the first contact one has with the ideas that will eventually be embodied in the program, Liskov and Berzins have emphasized the value of formal specifications. Their argument is that the increased rigor of a formal specification facilitates agreement among programmers on the meaning of the specification and, ultimately, the meaning of the program. The likelihood of such agreement is increased by the formal nature of the specifications, which should help limit the number and scope of possible interpretations.

More specifically, the precisely-defined syntax and semantics of a formal

---

[19] Gerhart and Yelowitz, p. 205.

[20] *Ibid.*

specification language should facilitate the construction of partial correctness proofs for the individual modules into which a program can be composed. This proof, combined with requirements for module termination, would constitute a verification of the program. Liskov and Berzins conclude their paper by expressing the current need for proof techniques for the various specification methods.

There are undeniable benefits from the use of formal program specifications; for example, the ability to decompose proofs of program properties, and the tendency of a formal statement to bring out details that, in an informal specification, might easily remain incompletely thought out and hidden under vague descriptions. It is important to *note* that increased formality in program specifications can decrease, but not eliminate, the inherent ambiguity of the early stage of development of a programming project. The informal nature of most existing specifications does succeed in presenting the main points in a fashion that is more understandable to most programmers than formal mathematical statements.[21] Liskov and Berzins recognize the role of informal specifications as a valuable and necessary complement to formal specifications. Their ideas can be taken as suggestions for improvement in the practice of software engineering. It is clear that a more responsible attitude towards program specifications should be nurtured, so as to eliminate a variety of bad programming habits (such as "the common habit of writing the specifications after writing the program"[22]) and the potentially dangerous effects they can lead to. Imprecise, loosely conceived design criteria are not likely to support comprehensible systems.

---

[21]Moreover - and this point cannot be overemphasized - most of the programs that I have been discussing all along are in a domain where *formal* specifications are impossible. In considering the universe of all programming applications, one must question how many of them are formalizable.

[22]Liskov and Berzins, p. 3.

The method of intermediate assertions, already discussed, is useful in breaking up the verification process into manageable units, and in alerting the programmer to particular areas of error (for example, the error must be in the code which lies between assertions 5 and 6). In addition, if the assertions are defined before coding, increased modularity is imposed; the goal is to divide a large system into a number of smaller, more manageable, and hopefully more easily comprehensible units. Elspas, *et. al.*, suggest "creating the assertions prior to formulating the details of the program algorithm, proving that the assertions correctly express the intent of the program, and, finally, writing the code that lies between the assertions."[23]

The difficulty of precisely defining the intent of a program notwithstanding,[24] the intermediate assertion approach seems to hold great promise for successful program verification. It has already affected the way we write software in a positive manner, by providing another method of evaluating the meaning of a program or program segment. "There is no doubt that many programs could benefit from an attempt to carry out an informal proof, at least for the fact that such a proof would reveal gross misunderstandings in the intended algorithm."[25]

An extension of the method of intermediate assertions is Wegbreit's scheme of program justifications. A justification is both a program assertion and a statement of how the path it applies to is to be proven correct. Justifications help clarify correctness proofs, but their real value is that they serve as additional text of the program itself, along with code and correctness specifications. In effect, the

---

[23] Elspas, *et. al.*, p. 142.

[24] I do not mean to dispense lightly with difficulties like this one, as I believe that they ultimately determine the extreme limitedness of proofs of correctness.

[25] Elspas, *et. al.*, p. 119.

programmer who utilizes this method proves the program correct while coding it.

Wegbreit's common sense ideas are instructive both in the study of verification and in the fulfillment of present programming tasks. In particular, he suggests "shifting part of the activity of program verification to *language design* . . . and to *programming practice*"[26] and contends that "program correctness is best achieved by explicitly considering the proof as part of the programming process."[27] Wegbreit calls for a change in the way programmers think about their work - an increased awareness that reliability is the responsibility of the programmer.

Other researchers have cautioned us not to abandon more "mundane" methods of program verification in favor of newly developed strategies that carry with them the legitimacy of mathematical formalism. The complex methods that researchers may find interesting are not always practical for programmers' uses.

> The kinds of algorithms that get 'proved' correct have nothing to do with software; given a choice between a very good algorithm with a *proof* of correctness, but which may be hard to understand, and a straightforward, unproved algorithm which an implementer believes he understands, the complex algorithm invariably loses. And it is the complex algorithms that are most interesting and have the most chance of being subjected to the sociology of '*proof*.'"[28]

In a similar spirit, Gerhart and Yelowitz remark that a common feature of program errors seems to be a "tendency to concentrate more effort on the harder parts which require sophisticated techniques and less effort on the 'obvious' and easier parts."[29] Sophistication should not overshadow thoroughness.

---

[26]Wegbreit, p. 31.

[27]*Ibid.*

[28]DeMillo, Lipton, and Perlis, p. 13.

[29]Gerhart and Yelowitz, p. 205.

A variety of good programming practices have evolved that in effect constitute informal efforts to verify software. For instance, through the years, there has been an increasing emphasis on debugging; now, well developed debugging tools are commonplace in most large programming projects. It is crucial to note that verification methods that have evolved from the practical experiences of programmers involve, first and foremost, an *expectation* of program errors.

It is clear that some common programming habits must be overcome if program verification techniques are to take hold, and that considerable effort will be required to do this successfully. "The methodologies proposed to increase software reliability are still in their early stages of development. The tasks are not easily taught or learned . . . ."[30] For instance, it is all too often the case that specifications and documentation are not regarded as integral parts of a programming project; yet, the effort that goes into these attempts to clearly state what a program is supposed to do is essential to the success of any verification process. Without this thoughtful planning and documentation of programming, reliable large scale systems are not possible. Until the mystique is removed from many people's concepts of programming and until programmers come to recognize the ethical (and not just the economic) value of dependable software, program verification will not be able to achieve its full potential.

6.1.3: Conclusions About Verification

Some common sense conclusions can be drawn from the preceding discussion of program verification. The first is really a lesson in humility. "We can want correctness in our programs, but we must settle for reliability."[31] Abstract, ideal

---

[30] *Ibid.*

[31] *Ibid.*, p. 206.

notions of correctness in terms of perfection are not likely to be fulfilled by real computer programs; this must be acknowledged in the course of making decisions about the use of "verified" programs. "We simply must learn to live with fallibility,"[32] and we must learn to do so responsibly, by not treating verified programs as perfect programs. ". . . in mature engineering disciplines, '*reliable*' never means '*perfect*' . . . engineers set probable limits of failure, relying on other design criteria to place these limits well above the conditions likely to be encountered in practice."[33] Current verification methods are best exercised with caution; that is, with an understanding that although they are helpful debugging tools, they do not guarantee correctness.

The second conclusion is that given a choice between informal and automatic methods of verification, moderation is most appropriate. "Experience with both . . . should convince us that neither type of evidence is sufficient and that both types are necessary."[34] Verification studies have much to offer in increasing our confidence in computer programs, but confidence is built up on many levels, and different kinds of verification evidence are needed. We come to believe things for many reasons; formally structured proofs, informal, intuitive explanations, trial and error, and insight all play important roles in inspiring trust in software. The best verification "package" will appeal to as many different channels of knowledge as possible.

Thirdly, computer scientists should begin to place more emphasis on the ethical issues that the use of any vital computer system brings into play. Software reliability is typically measured by the number of "bugs" encountered in a period of

---

[32] DeMillo, Lipton, and Perlis, pp. 16-17.

[33] Joyce, p. 61.

[34] *Ibid.*

time ("mean time between failures" or "mean time to recover"). We must decide in what situations these standards of reliability, and the systems that cannot be trusted beyond these limits, are acceptable. At times, we may want to question the basic criteria for the use of computers in a given application; the degree of vitality of the effects of a system, the reversibility of those effects, and the comprehensibility of the system are some issues that should be considered. At the very least, the dangers of overdependance on computer systems must be avoided by careful consideration of the ramifications of a dependant relationship between man and machine.

Lastly, I would go back to my first words about verification, at the beginning of this section, and state again that the goals of the field of program verification fundamentally miss the point. In a previous chapter, I discussed the misplaced use of computer systems as patch solutions to deep human difficulties. In the similar vein, verification proofs deal with only the superficial symptoms of system incomprehensible - program "mistakes."

We have every right to demand the highest standard of reliability from systems that we deal with, and to require justification for the use of a particular system; program verification plays a necessary and useful role in satisfying these demands. But it is wise to remember that a "proof" of correctness is not a license to establish a computer system as an autonomous decision making entity that is capable of initiating acts (but not capable of accepting responsibility for them). Unless we understand a system and are willing to accept responsibility for it, we should resist the temptations to depend on it.

6.2: Psychological Factors of Programming

My discussion in the preceding sections of this chapter has focused on the mechanical component of computer systems; Gerald Weinberg, among others, has emphasized nontechnical aspects of systems. The underlying message in his book, *The Psychology of Computer Programming*, is simple and obvious but nonetheless underemphasized: computer programming is fundamentally a human activity, and a lack of attention to the psychological aspects of this activity and to the social environment in which it takes place can result in the creation of computer programs which are undesirable from many viewpoints.

Early on, Weinberg acknowledges the impossibility of writing "perfect" computer programs and the need for computer programmers to recognize the limitations of their work: "Thus, there are degrees of meeting specifications - of 'working' - and evaluation of programs must take the type of imperfection into account."[35] The acceptable degree of conformance to formal program specifications should be made explicit in the course of designing a system. If the users of a system are made aware of the ways in which the actual system does and does not conform to the proposed specifications of system behavior, then they may make more informed decisions about their use of, dependance on, and trust in computer output.

One of the main concepts that Weinberg discusses is that of "egoless programming" - the training of software workers "to accept their humanity - their inability to function like a machine - and to value it and work with others so as to keep it under the kind of control needed if programming is to be successful . . . . the problem of the ego must be overcome by a restructuring of the social

---

[35]Gerald Weinberg, *The Psychology of Computer Programming* (New York: Van Nostrand Reinhold Company, 1971), p. 19.

environment and, through this means, a restructuring of the value system of the programmers in that environment."[36] The goal here is to debunk stereotypes that portray computer programmers as solitary workers whose expertise is best exercised in isolation (or, at least, in isolation from other people, though not necessarily from computers). Weinberg thinks that programmers should not be encouraged to identify themselves too personally with the programs they write,[37] since this can discourage cooperation between programmers, particularly in uncovering program "bugs" at an early stage of coding. Furthermore, Weinberg believes that egoless programming can result in faster average debugging time, more accurate estimates of the progress of a programming project as the work proceeds, and the generation of more reliable software. Weinberg continues to de-emphasize the role of the solitary (and potentially indispensable) programmer by stressing team involvement in setting goals. In addition, he advocates selecting workers who fit well within a shifting environment and are willing to work together.

It is clear that Weinberg's major point is that effective communication between the different members of a programming project and between system workers and users is necessary if we are to strive for better quality computer systems. This key issue is given relatively low status in a typical software project. For instance, documentation is frequently considered to be among the least important tasks associated with the generation of a new computer system, and certainly peripheral to the "real" work of designing and programming the system; instead, Weinberg believes that it should be elevated to a professional status of its own.

---

[36] *Ibid.*, pp. 56-57.

[37] This does not mean that programmers should not accept personal responsibility for their work, but rather that their identification with their programs should be on a professional, and not an overly emotional, level.

As is the case with most activities involving human beings, "what is needed in a programming project is slow, careful communication."[38] The social behavior of software workers may need to be modified to facilitate more productive communication between them.

6.3: Modern Programming Practices

Evidence, taken from existing computer applications, about the quality of current software is almost uniformly discouraging. Work in computer system reliability and program verification is, for the most part, still in the research stage of development, and interest in improving programming environments remains primarily academic.

> In spite of methodological improvements such as structured design and coding, chief programmer teams, on-line program development systems, high level languages and data base managers . . . , the delivered quality of large scale software, whether new or modified, remains disgraceful, except where the projects or the people involved are specially chosen. Pervasive cynicism about software is the justifiable consequence of the many situations where poor results follow long delays . . . . Technical panaceas have failed consistently in the past and promise to do so for the foreseeable future."[39]

Since software maintenance currently accounts for more than fifty percent (and sometimes as much as eighty percent) of a typical data processing budget,[40] there is obviously a strong incentive for the production of software that works. Why, then, is the quality of current computer software so poor?

The first thing to examine in attempting to answer this question is the goal of software that "works." Historically, this has been translated to mean nothing

---

[38] Weinberg, p. 109.

[39] D. H. McNeil, "Adopting a System Release Discipline," *Datamation*, Vol. 25, No. 1 (January, 1979), pp. 111-112.

[40] *Ibid.*, p. 112.

more than software that produces a tolerable approximation of the desired outputs. The predominant emphasis on minimizing the cost of computer system generation (for instance, incentive pay for early completion) has encouraged the satisfaction of the "workability" criterion through means of questionable reliability. Over-designing a first installation of a new system (building in – perhaps through some redundancy in the system – higher levels of flexibility, reliability, safety, etc. than may necessarily be needed, to handle unanticipated difficulties) is not immediately cost effective. Issues like dependability and comprehensibility are long term concerns not directly related to the market value of a system; because of this, computer manufacturers have often opted not to devote much of their energies to the refinement of failsafe and failure-proof techniques. Flexibility, modifiability, and maintainability have been treated as secondary components of quality assurance; primary components are those that relate to a system's immediate performance.

The emphasis in a computer system project has always been on coding the system software. Productivity indices typically measure lines of code written or number of compilable modules produced in a given time period. Structured programming, walk-throughs, and other modern programming practices are encouraging evidence of the current interest in software engineering, but this is still a fledgling discipline. Some programmers feel that the enforcement of these techniques succeeds only in rendering the activity of programming more inflexible by allowing no room for finding and cultivating one's own programming style. All too often, it appears that the motivating philosophy behind attempts to upgrade the quality of software calls for the implementation of techniques with low overhead and highly visible results, without much consideration for their long term value.

One difficulty is the single minded emphasis on improving the quality of computer programs, not systems as a whole. The achievement of better software must

involve changes in the operation of computer based organizations (software development and maintenance is a management issue, not a purely technical one) and in individual habits (*i.e.*, the development of a software engineering attitude and not merely the use of appropriate coding techniques). The continual change that is the way of life for most of the computer industry can be handled in more reliable ways. Software workers can avoid incremental patchwork on systems in production between scheduled releases of the system, managers can resist the pressures to have little improvements pasted onto the current release without going through a complete testing cycle, and users can recognize that they cannot demand new system features without paying for them with time and money.[41] The common themes in most current discussions of "the software problem" are the need for improved education, planning, and communication, involving managers, data processing analysts, programmers, operators, and users. Ultimately, producing better computer systems is a human problem. "In the long run the evolution of quality software depends upon people, not on systems analysis techniques, programming languages, or operating systems environments."[42]

6.4: Criticism

The most intriguing questions about incomprehensibility relate not merely to computer programs, but to computer systems and the social environments in which they exist.

> One can, with the same technology, design totally different outcomes by designing different social support systems . . . . The technology is the same, yet the pattern of use is highly dissimilar. The crucial decisions are sociological, not technological

---

[41] *Ibid.*, pp. 112, 114.

[42] *Ibid.*

> . . . . the . . . social questions frequently involve a difference if not a clash of values . . . . It is not a matter of right versus wrong but of right versus right. This is what makes moral decisions so difficult. And these are the hard questions.[43]

We have seen that there are personal and social effects of the use of computer systems, as well as technical effects. Lewis Mumford suggests reckoning up "the human disadvantages and costs, to say nothing of the dangers, of our unqualified acceptance of the system itself."[44]

Autonomous technology offers what it does "on one condition: that one must not . . . ask for [anything] that the system does not offer . . . . Once one opts for the system, no further choice remains."[45] In order to regain control over computer technique, we must continually criticize "the legitimacy of the technological question";[46] we must consciously decide whether or not this is what we want. The development of technology may be modified only if we widen our sphere of moral choices by considering other alternatives. This may be done in many ways; for instance, by utilizing the computer as only a partial solution to certain problems (". . . the computer too could be applied not simply wherever the opportunity arises, but only where it is deemed in the best interests of society"[47]) or by turning the positive aspects of computer applications to purposes other than the perpetuation of the technological system.

---

[43]Daniel Bell, "Hard Questions and Soft Minds: A Reply to Weizenbaum," Chapter 21 in *The Future Study on the Impact of Computers and Information Processing*, ed. by Michael L. Dertouzos (Cambridge, MA: The MIT Press, in press).

[44]Mumford., "Authoritarian and Democratic Technics," p. 57.

[45]*Ibid.*

[46]Joseph Weizenbaum, "On the Impact of the Computer on Society," *Science*, 176 (May 12, 1972), p. 612.

[47]Abbe Mowshowitz, *The Conquest of Will: Information Processing in Human Affairs* (USA: Addison-Wesley Publishing Company, 1976), p. 62.

> The very leisure that the machine now gives . . . can be profitably used, not for further commitment to still other kinds of machine, furnishing automatic recreation, but by doing significant forms of work, unprofitable or technically impossible under mass production: work dependant upon special skill, knowledge, aesthetic sense.[48]

Even in a technological world, it is possible to criticize the system, but we forfeit our right to criticize if we allow technology to dictate its own course.

6.5: Responsibility

The necessity for a strong sense of individual moral responsibility in today's society cannot be overemphasized. One starting point for computer professionals might be the set of "Guidelines for Professional Conduct in Information Processing" set forth by the Association for Computing Machinery; in the preamble, the ACM urges the following:

> The professional person, to uphold and advance the honor, dignity and effectiveness of the profession in the arts and sciences of information processing, and in keeping with high standards of competence and ethical conduct: Will be honest, forthright and impartial; will serve with loyalty his employer, clients and the public; will strive to increase the competence and prestige of the profession; will use his special knowledge and skill *for the advancement of human welfare.*[49] [my emphasis]

Responsibility is here defined not merely in a narrow professional sense, but rather in a broad sense that takes into account the relationships between information processing professionals and employers, clients, other professionals, and the public. What requires additional thought is the meaning of responsibility in terms of possible repercussions on, for instance, programmers. Questions as to whether or not

---

[48] Mumford, "Authoritarian and Democratic Technics," p. 58.

[49] Reprinted in C. C. Gottlieb and A. Borodin, *Social Issues in Computing* (New York: Academic Press, 1973), pp. 236-237.

computer professionals may be sued or put in jail (for example) for the production of "unacceptable" computer systems ought to be considered.

The most important step in regaining control over technology is the recognition that we alone bear responsibility for our technological systems, and we are bound - in technological endeavors no less than in other endeavors - by the social ethics governing human behavior in society. The employment of computer systems in tasks involving people involves a social decision. The field of computer science enjoys substantial public support and has significant impacts on society; it has a social responsibility. Moreover, this social responsibility cannot be denied - it is exercised by default, through tacit approval of new technological systems, even if explicit judgments are not made.

> . . . the underlying goal of science and technology is to improve 'the quality of life.' This implies that those closest to the technology have a special *obligation* to question the uses and consequences of their work, to exert as much beneficial influence as possible, to direct technological development and its application, and even to refuse to work on projects deemed not socially constructive.[50]

In questioning our technology, we must critically examine the particular role we play in the system. ". . . only the leader who is ready to step down has a real chance of success."[51] Technological inevitability need not be accepted; individual decision making is possible, and constitutes a necessary step towards a return to a human-centered technology. "It is possible, given courage and insight, for man to deny technology the prerogative to formulate man's questions. It is possible to ask human questions and to find humane answers."[52]

---

[50]Gottlieb and Borodin, pp. 241-242.

[51]Weinberg, p. 85.

[52]Joseph Weizenbaum, "On the Impact of the Computer on Society," *Science*, 176 (May 12, 1972), p. 614.

6.6: Humanism

It should be clear by now that the improvement of computer programming techniques is a significant area of current computer science research (and, to a much lesser extent, of present computer science practice), but that the emphasis of this chapter has moved to the improvement of humanizing "techniques" - actually, attitudes - to apply to the creation and use of computer systems.

In current practice, humanizing a computer system is often done cosmetically; for instance, by having the system communicate with users in a soothing, English-like dialect. While this is helpful, meaningful humanization of computer systems must go beyond superficialities; humanizing a system must involve injecting the human element into a technological system.

> . . . we had better map out a more positive course: namely, the reconstitution of both our science and our technics in such a fashion as to insert the rejected parts of the human personality at every stage in the process. This means gladly sacrificing mere quantity in order to restore qualitative choice, shifting the seat of authority from the mechanical collective to the human personality and the autonomous group, favoring variety and ecological complexity instead of stressing undue uniformity and standardization, above all, reducing the insensate drive to extend the system itself, instead of containing it within definite human limits . . . . We must ask, not what is good for science or technology, . . . , but what is good for man . . . ."[53]

At the very least, humanization requires a recognition of human values. Computers may be used to explore alternative courses of action, but values must be included in decision making about the use of computerized "answers." "What is evidently wanted is a set of balance sheets in which the relative merits of each solution to a technological problem are analyzed both on technological grounds such as

---

[53] Mumford, "Authoritarian and Democratic Technics," p. 58.

safety, ease of operation, complexity, and esthetics, and on ethical grounds such as moral considerations, effects on the quality of human life, liberty and dignity, and other human values."[54] We must come to believe that the dictates of human beings supersede those of technological systems; there is relatively little compelling evidence that modern society does believe this. In addition, a strong sense of control over the use of technology should be nurtured; if necessary, this should involve "cut[ting] the whole system back to a point at which it will permit human alternatives, human interventions, and human destinations for entirely different purposes from those of the system itself."[55]

In an organizational context, care must be taken to avoid undesirable effects of the increased rigidity, formality, and alienation that frequently accompany computerization. One should not ignore the nonrational aspects of social conduct by administering a social organization according to purely technological criteria. The modern bureaucracy is largely irresponsible in its pursuit of efficiency - indifferent to human needs and unsupportive of the promotion of interaction and communication across hierarchical levels. Mowshowitz points out the difficulty of challenging the status quo: ". . . one critical feature is a mechanism for establishing a continuing dialogue between managers and workers or ordinary citizens. This is not likely to emerge spontaneously, since it presupposes a fundamental shift in values from productivity and efficiency to human well-being."[56] Two things are especially important in what Mowshowitz and others have said about improving conditions in a technological society. First, communication is seen as a key factor in changing the existing situation; one aspect of this is that more of the people who interact with computer

---

[54]Mowshowitz, pp. 271-272.

[55]Mumford, "Authoritarian and Democratic Technics," p. 59.

[56]Mowshowitz, p. 201.

systems should have access to information about the system, as well as the power to use this information. Second, perhaps the ultimate deterrent to comprehending computer systems is the human value system that places such a premium on technology.

Finally, and most importantly, technological system incomprehensibility is a human problem, and the most significant and difficult questions that it raises should be answerable to humanistic concerns. Any meaningful comprehension of our interactions with computer systems must be preceded by a better understanding of our own role in a technological society, and this in turn requires an understanding of interpersonal difficulties, human priorities, and ethical values.

> What should this teach us, particularly with respect to the question of at least preserving if not enhancing human choice in human affairs?
>
> Certainly that the construction of reliable computer software awaits, *not so much* results of research in computer science, but rather a deeper theoretical understanding of the human condition.[67]

Before computer systems can be made truly comprehensible, human systems must be better understood. Before we can control our technological systems, we must learn to value people more than technics.

---

[67] Joseph Weizenbaum, "Human Choice in the Interstices of the Megamachine," p. 14. Lecture presented at the IFIPS Conference on "Human Choice and Computers," Vienna, Austria, in June, 1979.

This empty page was substituted for a blank page in the original document.

VITAE

Paul Armer: computer and information scientist. Armer is presently a research associate in the area of programming technology and society at Harvard University. Previously, he headed the Computer Science department at the Rand Corporation. His research interest is the impact of technology on society in the application of computers as information processors.

Daniel Bell: sociologist. Bell has taught sociology at the University of Chicago, Columbia University, and Harvard University. He is a contributor to a variety of academic and technical journals, and has served on the editorial staffs of *Fortune* (1948-58), *Daedalus*, *The American Scholar*, and *The Public Interest*. Bell is a former member of the President's Commission on Technology, Automation and Economic Progress, fellowship recipient of the Center for Advanced Studies in Behavioral Sciences, and winner of an American Council on Education prize. He is the author of several books, most recently *The Cultural Contradictions of Capitalism* (1976).

Valdis Berzins: computer scientist. Berzins is a PhD graduate of MIT's computer science department, and is presently involved in research with the computation structures group there.

Alan Borodin: Borodin is presently a faculty member in the department of computer science at the University of Toronto.

Kenneth Mark Colby: psychiatrist and computer scientist. Colby has taught psychology at Stanford University. Presently, he is a senior research associate in computer science at Stanford; in addition he is a career scientist for the National Institute of Mental Health. His research deals with computer simulation of belief systems. Colby's publications include *Computer Models of Thought and Language* (1973).

Richard A. DeMillo: mathematician and computer scientist. DeMillo has worked as a research assistant at the Los Alamos Science Laboratory of the University of California and as a research associate at the Georgia Institute of Technology; presently, he is a computer science faculty member at the University of Wisconsin. Research interests - theory of computation, theory of programming languages, program language design.

Daniel C. Dennett: philosopher. Dennett received his education at Harvard University and at Oxford (D. Phil.). He has served on the faculties of philosophy departments at the University of California, Harvard University, the University of Pittsburgh, and Tufts University, where he has been chairman of the department since 1976. Dennett has been involved in a large number of colloquia and conferences and is the author of numerous articles, among them the following on artificial intelligence: "Artificial Intelligence: Limitations and Implausibilities" (1978), "Badmouthing AI" (1978), "Artificial Intelligence as Philosophy and as Psychology" (1979), and "Philosophical Issues in Artificial Intelligence" (1979).

Bernard Elspas: Elspas is currently a computer scientist at the Stanford Research Institute, California.

Jacques Ellul: Ellul was born in Bordeaux, France in 1912. He studied at the University of Bordeaux and at the University of Paris, receiving a doctorate in law from the latter. He has been Professor of the history of law at the University of Bordeaux since 1946. Professor Ellul's European reputation is immense, and his reputation in America has been firmly established by the publication here of several books, including the following: *Propaganda: The Formation of Men's Attitudes* (1965), *The Political Illusion* (1965), *A Critique of the New Commonplaces* (1968), and *The Autopsy of Revolution* (1971).

Robert Fano: educator. Fano holds a PhD in electrical engineering from MIT, and has worked in MIT's electrical engineering and computer science department, Research Lab of Electronics, Radiation Lab, and Lincoln Laboratory. He has been director of MIT's Project MAC and has served as associate head for computer science and engineering in the EECS department. He is now the Ford Professor of Electrical Engineering and Computer Science at MIT.

Lawrence Flon: computer scientist. Flon has worked in the department of computer science at Carnegie-Mellon and Texas University, and at Xerox Palo Alto Research Center. His current research is in programming languages, program correctness, and program design.

Susan L. Gerhart: Gerhart holds academic degrees in mathematics (BA), communication sciences (MS) and computer science (PhD). She was previously a visiting assistant professor of computer science at the University of Toronto, and is now an assistant professor of computer science at Duke University, North Carolina. She is also employed as a senior software engineer at SofTech, Massachusetts, and as a visiting scientist at ICASE, Virginia. Research – program verification, including testing, the theory and practice of proving correctness of programs, and programming methodologies which reduce the difficulties of verification.

C. C. Gottlieb: Gottlieb is a faculty member in the department of computer science at the University of Toronto.

Jeanne Erard Gullahorn: psychologist. Gullahorn is presently a faculty member at Michigan State University; she also acts as a consultant for System Development Corp., California. She has published several articles, including "Computer simulation of role conflict resolution." Research – computer simulation of social behavior, psychology of women, and cross-cultural research.

John Taylor Gullahorn: sociologist. Gullahorn is currently a faculty member in sociology and computer instruction at Michigan State University; he also works as a consultant for the U.S. State Department and numerous others. His research interests include the use of computers to test and develop theories of social behavior.

Arie Nicholas Habermann: computer scientist. Habermann has been a faculty member in the Netherlands and at a variety of European universities, and now teaches at Carnegie-Mellon University, in addition to working as a consultant for Westinghouse Corporation and DEC. Research - programming languages, programming systems, operating systems.

John Kemeny: college president. Kemeny worked on the Los Alamos Project and at the Institute for Advanced Study (for Einstein). He has taught at the Institute for Advanced Study and at Dartmouth; presently, he is president of Dartmouth University. Kemeny is a member of numerous professional associations and committees (including HEW and NRC committees), and author of numerous books about mathematics, programming, and science. His research interests are in mathematical models for the social sciences and the philosophy of science.

Philip Kraft: computer scientist. Kraft currently teaches sociology at SUNY in Binghampton, New York. His research deals with class relations of societies in general and the way they are played out in the development of technology. His present work is with representative of trade unions in the area of political implications of the widespread introduction of microprocessor technology. Another of Kraft's current interests is in the role of women and minority men in the field of computing.

Butler Wright Lampson: computer scientist. Lampson has served on the faculty of the University of California at Berkeley, and now works as a research fellow in computer science at Xerox Palo Alto Research Center, in addition to directing system development at Berkeley Computer Corp. Research - operating systems and programming languages.

John H. Lehman: Lehman's background is in computer software development, and he has worked with computer systems in a variety of roles, from programmer to project manager. He held a variety of data processing positions while in the Air Force; at different times, he has also held staff positions in planning, policy formation, and requirements determination. He is presently a computer consultant.

Richard J. Lipton: Lipton has been a faculty member in computer science at Stanford University and Yale University; presently, he teaches at Berkeley University. His research is in data structures, algorithms, and computational complexity.

Barbara Liskov: computer scientist. Liskov has worked at the MITRE Corp. and in Carnegie-Mellon University's department of computer science. She is currently a computer science faculty member at MIT, where she heads the computation structures research group. Her research interests are the design of programming languages and software development.

Zohar Manna: computer scientist. Manna has been associated with the Stanford Research Institute and Stanford University's department of computer science. His research is in the areas of program correctness, program schemas, algorithms, and mathematical logic.

Bruce Mazlish: educator, historian. Mazlish, a PhD graduate of Columbia University, has been an instructor at the University of Maine, Columbia University, and MIT, and has been a faculty member and chairman of the humanities department at MIT. He has served on panels for the National Science Foundation and the American Academy of Arts and Sciences. He is the author of numerous books; most recently (1976), *The Revolutionary Ascetic: Kissinger, The European Mind in American Policty.*

D. H. McNeil: computer scientist. McNeil is now an independent consultant in the Philadelphia area. He has worked as a computer programmer, system analyst and designer, instructor for IBM's general systems division, and technical advisor for Shared Medical Systems (where he developed many of the concepts of a system release discipline).

Marvin Minsky: mathematician. Minsky holds academic degrees from the Phillips Andover Academy, Harvard, and Princeton (PhD). He is one of the founders of MIT's artificial intelligence laboratory and served as director of that lab; he has also served on the faculty of the mathematics and computer science departments at MIT (where he is currently Donner Professor of Science). Author of several books, Minsky was the recipient of the 1970 Turing Award. His research is in the areas of artificial intelligence, theory of computation, psychology, and engineering.

Abbe Mowshowitz: Mowshowitz is an associate professor of computer science at the University of British Columbia and a visiting research associate in the department of computer science at Cornell University. He has special interests in combinatorial mathematics and graph theory, in addition to the social impact of technology.

Lewis Mumford: author. Mumford has served on the faculties of Stanford University, University of Pennsylvania., MIT, and the University of California at Berkeley, and was a senior fellow at Wesleyan University's Center for Advanced Studies. He has been the recipient of numerous award; among them are the National Book Award (1962), the US Medal of Freedom (1964), the Emerson Thoreau Medal of the American Academy of Arts and Sciences (1965), the Gold Medal for Belles Lettres of the National Institute of Arts and Letters (1970), and the National Medal for Literature (1972). He has been a Fellow with numerous professional organizations, is an honorary member of *Phi Beta Kappa*, and is a decorated Knight of the British Empire (honorary). He is the author of numerous books, including *The Myth and the Machine* (1967, 1970 - 2 volumes) and *Technics and Civilization* (1934).

Allen Newell: educator. Newell holds academic degrees in physics and industrial administration. He has served on the faculty of Carnegie-Mellon University and was a consultant with the Rand Corp. and with the Xerox Corp. He belongs to a variety of professional organization, has received several professional awards (including the 1975 Turing Award, which he shared with Herbert Simon), and is the author of a number of books (including *Human Problem Solving*, written with H. Simon). Research – artificial intelligence, psychology, programming systems, and computer structures.

Robert L. Patrick: Patrick is presently a freelance computer specialist based in southern California. His clients include companies with products in aerospace, computing, finance, and manufacturing, and he has performed 28 audits of computer centers. He is the author of the 1974 AFIPS Security Manual, as well as many books and reports, including the 1978 NBS study on data integrity practices.

Edwin W. Paxson: Paxson currently works for the Rand Corp. in California.

Alan J. Perlis: educator, computer scientist. Perlis has academic degrees from Carnegie-Mellon Institute of Technology, California Institute of Technology, and MIT (PhD in mathematics); he now serves as chairman of the computer science department at Yale University. Perlis is a past president of the Association of Computing Machinery (ACM), past editor-in-chief of Communications of the ACM, member of numerous professional organizations. He has served on numerous committees, including those for the National Institute of Health and the National Science Foundation.

Theodore Roszak: author, editor, educator. Roszak is a former Guggenheim Fellow (1971-72) and is the author of numerous books, including *Sources: An Anthology of Contemporary Materials Useful for Preserving Sanity While Braving the Great Technological Wilderness* (1972).

Thomas Brown Sheridan: Sheridan is currently a professor of mechanical engineering at MIT; in addition, he acts as a consultant to various coporations. His research interests include group decision technology, mathematical models of the human operator in control systems, and man-machine interations.

Herbert Alexander Simon: social scientist. Simon has served on the faculty and administration of the University of California, the Illinois Institute of Technology, and Carnegie-Mellon University (where he is now a trustee). His numerous awards include the 1975 Turing Award (shared with A. Newell) and the 1978 Nobel Prize in Economics. He is a member of numerous professional organizations and author of several books, including *The Sciences of the Artificial* (1968).

T. D. Sterling: Sterling is currently associated with the department of computer science at Simon Fraser University, British Columbia.

Howard K. Thompson, Jr.: computer scientist, biomathematician. Thompson earned an MD from Columbia University; presently, he is a professor of medicine at Baylor College medical school. Research - computers in medicine.

Alan Turing: Turing was a British mathematician who developed the idea of an abstract computing machine that can carry out a certain class of computing tasks. This machine, known as a Turing machine, is now fundamental to the study of theoretical computer science. Turing was involved in significant cryptography work during World War II. Turing published a number of articles on intelligent machinery and man as machine.

Ben Wegbreit: Wegbreit is associated with the Xerox Palo Alto Research Center.

Richard Waldinger: computer scientist. Waldinger has pursued research in mathematics and artificial intelligence; presently, he works at the Stanford Research Institute. Research - artificial intelligence, automatic program synthesis and verification, mechanical theorem proving, and robotics.

Gerald M. Weinberg: computer scientist. Weinberg is professor of computer systems at the School of Advanced Technology, State University of New York. He has been involved with various aspects of computer technology since 1956, including software implementation and both hardware and software design. Dr. Weinberg has written a number of books on computer programming, plus several dozen articles on computing, problem solving and systems theory. A member of ACM and AAAS, he is also a member, as well as former Secretary, of the Society for General Systems Research.

Joseph Weizenbaum: Weizenbaum received his higher education at Wayne State University. He has worked on the development of a variety of computer systems with Wayne State University, Computer Control Co., Bendix Computer Division, and the GE Computer Department (where he helped develop programming systems to aid in the design of the computer system which was to be built for the Bank of America). He has held academic appointments with Stanford University, Harvard University, the Technical Institute of Berlin, and MIT (where he is presently a professor of computer science). He is a member of numerous professional and political organizations, and author of many articles as well as the book, *Computer Power and Human Reason: From Judgment to Calculation* (1972). Once known mainly for computer science applications such as ELIZA (a natural language processing system), Weizenbaum's interests have shifted in recent years to the area of societal implications of computers; in particular, to the issues of responsibility, incomprehensibility, technological inevitability, and the implications of trusting computer systems.

Harvey Wheeler, Jr.: political scientist. Wheeler has taught at John Hopkins University and at Washington and Lee. He is currently a senior fellow with the Center for the Study of Democratic Institutions in California and a freelance consultant. His research is in the area of political theory.

Norbert Wiener: Wiener studied logic with Bertrand Russell and was on the faculty of MIT's mathematics department. He demonstrated an early interest in analogies between electronic and biological devices. Wiener's belief that there was an essential unity of problems centering around communication and control - in machine or living tissue - led to his foundation of the science of cybernetics.

Langdon Winner: Winner is a writer, teacher, and sometimes music critic. He has worked for Rolling Stone magazine, the Pentagon, and the M.I.T. He is the author of *Autonomous Technology: Technics-out-of-control as a theme in political thought* and numerous articles on technology and the human condition. Mr. Winner is currently writing a book on technological design and the quality of public life.

Lawrence Yelowitz: Yelowitz holds academic degrees in mathematics (BS and MAT) and computer science (PhD). He has been involved with industrial and governmental computer programming applications, as well as research work with IBM and the AI group at the National Institute of Health. He has taught computer science at New Mexico Institute of Mining and Technology and at the University of California at Irvine, and is now an assistant professor of computer science at the University of Pittsburgh. A member of several professional organizations, Yelowitz was the first recipient of the Samuel N. Alexander Memorial Award sponsored by the Washington, D.C. chapter of ACM. Yelowitz's publications are in the area of program correctness. Research - program correctness, operating systems, and data structures.

The information in this section was compiled from the card catalog of MIT's computer science reading room, and from the following sources:

Who's Who in America, 40th edition (1978-79). (Illinois: Marquis Who's Who, Inc.).

American Men and Women of Science, 13th edition. Jaques Cattell Press, ed. (New York: R. R. Bowker Company, 1976).

American Men and Women of Science, 12th edition, *The Social and Biological Sciences*. Jaques Cattell Press, ed. (New York: R. R. Bowker Company, 1973).

American Men and Women of Science, 12th edition, *Physial and Biological Sciences*. Jaques Cattell Press, ed. (New York: R. R. Bowker Company, 1972).

This empty page was substituted for a blank page in the original document.

BIBLIOGRAPHY

Armer, Paul. "Attitudes Toward Intelligent Machines." In *Computers and Thought*. Edited by Edward A. Feigenbaum and Julian Feldman. USA: McGraw-Hill, Inc., 1963.

Bell, Daniel. "Hard Questions and Soft Minds: A Reply to Weizenbaum." Chapter 21 in *The Future Study on the Impact of Computers and Information Processing*. Edited by Michael L. Dertouzos. Cambridge, MA: The MIT Press, in press.

Colby, Kenneth Mark. "Computer Psychotherapists." UCLA Department of Psychiatry, Algorithmic Laboratory of Higher Mental Functions Memo ALHMF-14.

Darrow, Joel W. and Bellilove, James R. "Keeping Informed." *Harvard Business Review*, 56, No. 6 (November-December, 1978), 180-194.

Dembart, Lee. "Experts Argue Whether Computers Could Reason, and if They Should." *New York Times*, May 8, 1977.

DeMillo, Richard A., Lipton, Richard S., and Perlis, Alan J. "Social Processes and Proofs of Theorems and Programs." Yale University Department of Computer Science Research Report #82 (1976).

Denison, D. C. "The cybernetic revolution." *Boston Phoenix*, September 26, 1979.

Dennett, Daniel C. *Brainstorms - Philsophical Essays on Mind and Psychology*. USA: Bradford Books, 1978.

Denning, Peter J. "On Being One's Own Programming Self." AFIPS National Computer Conference, 1977, p. 283.

Ellul, Jacques. *The Technological Society*. New York: Vintage Books, 1964.

Elspas, Bernard, *et. al.* "An Assessment of Techniques for Proving Programs Correct." *Computing Surveys*, Vol. 4, No. 2 (June, 1972), pp. 97-147.

Fano, Robert. "On the Social Role of Computer Communications." *Proceedings of the IEEE*, 60, No. 11 (November, 1972), 1249-1253.

_____. "The Computer Utility and the Community." *1967 IEEE International Convention Record*, Part 12, pp. 30-37.

Flon, L. and Habermann, A. N. "Towards the Construction of Verifiable Software Systems."

Franklin, James L. "Technologists warn of need for controls." *Boston Globe*, July 20, 1979.

Gerhart, Susan L. and Yelowitz, Lawrence. "Observations of Fallibility in Applications of Modern Programming Methodologies." *IEEE Transactions on Software Engineering*, SE-2, No. 3 (September, 1976), 195-207.

Gottlieb, C. C. and Borodin, A. *Social Issues in Computing*. New York: Academic Press, 1973.

Gullahorn, John T. and Gullahorn, Jeanne E. "A Model of Elementary Social Behavior." In *Computers and Thought*. Edited by Edward A. Feigenbaum and Julian Feldman. USA: McGraw-Hill, Inc., 1963.

Hapgood, Fred. "The Myth of Computer Control." *Free Paper*, November 4, 1978.

Joyce, James. "Human Factors in Software Engineering." *The First West Coast Computer Faire: Conference Proceedings*. Edited by John C. Warren, Jr. Palo Alto, CA: Computer Faire, 1977.

Kemeny, John G. *Man and the Computer*. New York: Charles Scribner's Sons, 1972.

Kent, William. *Data and Reality*. New York: North-Holland Publishing Company, 1978.

Kinnucan, Paul. "Automation Comes to the White House." *Mini-Micro Systems*, May, 1979, pp. 98-103.

Kraft, Philip. *Programmers and Managers: The Routinization of Computer Programming in the United States*. New York: Springer-Verlag, 1977.

Lampson, Butler W. "Building Programs." MIT Laboratory for Computer Science Distinguished Lecturer Series. May 1, 1979.

Laudon, Kenneth C. Review of *The Conquest of Will: Information Processing in Human Affairs*, by Abbe Mowshowitz. In *Science*, 195, September, 1976, 1111-1112.

Lehman, John H. "How Software Projects Are Really Managed." *Datamation*, January, 1979, 25, No. 1, 119-129.

Liskov, Barbara H. and Berzins, Valdis. "An Appraisal of Program Specifications." MIT Laboratory for Computer Science CSG Memo 141 (July, 1976).

Lynch, Mitchell C. "A Computer Error: Trying to Use One in Your Home." *Wall Street Journal*, May 14, 1979.

Manna, Zohar, and Waldinger, Richard. "The Logic of Computer Programming." Stanford AI Lab Memo AIM-288 (August, 1977).

Mazlish, Bruce. "The Fourth Discontinuity." In Adams, J. Mack, and Haden, Douglas H. *Social Effects of Computer Use and Misuse*. New York: John Wiley & Sons, 1976.

McCorduck, Pamela. *Machines Who Think.* San Francisco: W. H. Freeman and Company, 1979.

McNeil, D. H. "Adopting a System Release Discipline." *Datamation* (January, 1979), Vol. 25, No. 1, pp. 110-117.

Minsky, Marvin. "Steps Towards Artificial Intelligence." In *Computers and Thought.* Edited by Edward A. Feigenbaum and Julian Feldman. USA: McGraw-Hill Book Company, Inc., 1963.

Mowshowitz, Abbe. *The Conquest of Will: Information Processing in Human Affairs.* USA: Addison-Wesley Publishing Company, 1976.

Mumford, Lewis. "Authoritarian and Democratic Technics." In *Technology and Culture.* Edited by Melvin Kranzberg and William H. Davenport. New York: Schocken Books, 1972.

_____. "Technics and the Nature of Man." In *Technology and Culture.* Edited by Melvin Kranzberg and William H. Davenport. New York: Schocken Books, 1972.

Newell, Allen and Simon, H. A. "GPS, A Program That Simulates Human Thought." In *Computers and Thought.* Edited by Edward A. Feigenbaum and Julian Feldman. NY: McGraw-Hill, Inc., 1963.

Patrick, Robert L. "Things Are Looking Up For Software." *Datamation,* January, 1979, 25, No. 1, 131-133.

Paxson, Edwin W. "Computers and National Security." In *Computers and the Problems of Society,* ch. 3. Edited by Harold Sackman and Harold Borko. New Jersey: AFIPS Press, 1972.

Reineman, Laurence, "Computers and the Workplace, MIT lecture series sponsored by the Program for Science, Technology, and Society, March 7, 1979.

Rickover, Vice-Admiral H. G. "A Humanistic Technology." In *Technology and Society.* Edited by Noel deNevers. USA: Addison-Wesley Publishing Company, 1972.

Roszak, Theodore. "The Computer - A Little Lower Than the Angels," *The Nation,* 222 (May 1, 1976), 533-536. Review of *Computer Power and Human Reason,* by Joseph Weizenbaum.

Shenker, Israel. "Man and Machine Match Minds at MIT." *New York Times,* August 27, 1977.

Sheridan, Thomas B. and Ferrell, William R. *Man-Machine Systems: Information, Control, and Decision Models of Human Performance.* Cambridge, MA: The MIT Press, 1974.

Shrobe, Howard E., Rich, Charles, and Waters, Richard C. "Computer Aided Evolutionary Design for Software Engineering." Massachusetts Institute of Technology Artificial Intelligence Laboratory A. I. Memo 506, January, 1979.

Simon, Herbert A. "What Computers Mean For Man and Society." *Science*, 195 (March 18, 1977), 1186-1191.

Sterling, T. D. "Consumer Difficulties With Computerized Transactions: An Empirical Investigation." *Communications of the ACM*, 22, No. 5 (May, 1979), 283-289.

"The Computer Society." *Time*, 111, No. 8 (February 20, 1978), 44-59.

Thompson, Howard. *Joint Man/Machine Decisions*. Cleveland, Ohio: Systems Procedures Assoc., 1965.

Turing, A. M. "Computing Machinery and Intelligence." Reprinted from *Mind: A Quarterly Review of Psychology and Philosophy*, Vol. 59 (1950), pp. 236 ff. In Adams, J. Mack, and Haden, Douglas H. *Social Effects of Computer Use and Misuse*. New York: John Wiley & Sons, 1976.

Wegbreit, Ben. "Constructive Methods in Program Verification." Xerox Palo Alto Research Center (December, 1976).

Weinberg, Alan M. "Can Technology Replace Social Engineering?" In *Technology and Society*. Edited by Noel deNevers. USA: Addison-Wesley Publishing Company, 1972.

Weinberg, Gerald. *The Psychology of Computer Programming*. New York: Van Nostrand Reinhold Company, 1971.

Weizenbaum, Joseph. "Apollo Agonistes." Lecture presented at SUNY in Albany, New York. April 20, 1979.

_____. *Computer Power and Human Reason: From Judgment to Calculation*. New York: W. H. Freeman, 1976.

_____. "Controversies and Responsibilities." *Datamation* (November 15, 1979), pp. 173-179.

_____. "Human Choice in the Interstices of the Megamachine." Lecture presented at the IFIP Conference on "Human Choice and Computers." Vienna, Austria: June, 1979.

_____. "On the Impact of the Computer on Society." *Science*, 176 (May 12, 1972), 609-614.

_____. "Once More: The Computer Revolution." February 27, 1978. In *The Computer Age: A Twenty-Year View*. Edited by Michael L. Dertouzos and Joel Moses. Cambridge, MA: The MIT Press, in press.

_____. "Technological Detoxification." World Council of Churches Conference on Faith, Science, and the Future, at MIT July 18, 1979.

_____. "The Last Dream." *The Conference Board Magazine*, XIV, No. 7 (July, 1977), 34-46.

_____. "The Two Cultures of the Computer Age." *Technology Review*, 71 (April, 1969), 54-57.

Wheeler, Harvey. "Artificial Reasoning Machines and Society." In *Computers and the Problems of Society*, ch. 13. Edited by Harold Sackman and Harold Borko. New Jersey: AFIPS Press, 1972.

Wiener, Norbert. *God and Golem, Inc.* Cambridge, MA: The MIT Press, 1964.

Winner, Langdon. *Autonomous Technology: Technics-out-of-control as a theme in political thought.* Cambridge, MA: The MIT Press, 1977.