6

An Ecological Approach to Responsible Systems Development*

Reinhard Keil-Slawik
Technische Universitaet Berlin

INTRODUCTION

Having been politically active as a student at the Technical University of Berlin in the mid-seventies , I felt a strong need to take account of the problems of computer use and misuse in my daily work as a computer professional when I started my university career. I began looking for practical and theoretical ways to achieve this.

Together with some of my colleagues I founded a *Wissenschaftsladen* (Science Shop). This was open to anyone in search of technical or scientific advice, provided that this related to his or her own personal needs, and was not connected with commercial goals. For instance, we advised several "alternative technology projects" on whether to buy a computer or not; and shop stewards and employees who where faced with the installation of a computer system they didn't want.

This change of social perspective, from the developer's to the user's point of view, enabled us to learn a great deal about the potential of computers and their everyday use (Beuschel, Bickenbach, & Keil, 1983); about the problems of limiting the size of a project, with or without the use of high technology; about the wide variety of possible applications of one and the same DP system (Beuschel, Bickenbach, & Keil, 1983). And we found we were able to provide constructive advice and concrete help in a great many cases. However, this changing of social perspectives proved to be too time-consuming in the long run.

New technological developments seem to produce problems more rapidly than they can be solved—if they can be solved at all. What we need, then, is a closer approximation of cause and effect, that is, we should try to preclude at

* I would like to thank Christiane Floyd, Friedrich Holl, and Dirk Siefkes for their valuable comments on my manuscript. Furthermore, I am greatly indebted to Phil Bacon for polishing up the text idiomatically and stylistically.

least some of the problems instead of trying to "repair" them afterwards. Within the bounds of the conventional scientific approach, however, it has proved impossible to bring science and policy making together.

On a theoretical level, I began to study the historical role of science in society in general, and the history of the computer in particular. I discovered that the reductionist approach to scientific inquiry was identifiable as one of the main problem sources. A new scientific approach was needed. That was more easily said than done. Criticizing the conventional approach was easier than developing a new approach which would avoid all the problems, shortcomings, and limitations inherent in reductionist thinking. It took several years of working in various organizations and on different topics, ranging from highly philosophical questions to purely technical ones, before the idea of an ecological approach took shape in my mind. And still this process continues in a highly dynamic fashion.

In this chapter, my main concern is to summarize what has been achieved and to initiate a discussion as to how the ecological approach can be improved so as to compete with the conventional approach on all appropriate levels of scientific enquiry. I hope to encourage more computer professionals to work along the same lines.

The following section presents some arguments relating to the problems inherent in reductionist thinking. A subsequent section goes on to plead for an enrichment of the conventional scientific approach by an ecological perspective, which would allow us to consider together different phenomena separated by reductionist thinking. Basic ideas which I judge to be essential will be briefly outlined; these ideas will be taken up in the next section and illustrated in the context of software development. The limitations of the reductionist approach and the opportunities offered by the adoption of an ecological perspective are discussed with reference to the notion of open/closed systems. This section also presents our approach to software development, elaborated at the Technical University of Berlin. The next section attempts to provide some ethical criteria for responsible systems development, which are strongly connected with the ecological perspective presented here. The final section looks at the potential for future work as well as the limitations of the ecological approach.

REDUCTIONISM CONSIDERED DOUBTFUL

What we today call reductionist thinking has its roots in the emergence of modern science and can be found at all levels of scientific activity, at the institutional as well as the methodological level.

The mid-17th century saw the founding of the "Royal Society" (London, 1662) and the "Acadmie Royale des Sciences" (Paris, 1666). The far-reaching privileges attained by these societies from the monarchist state were paid for by the strict separation of science and politics. Though scientists worked directly for

military and economic ends (see Berman 1981, and Bernal, 1965), scientific enquiry claimed to rely purely on science-immanent criteria, as formulated by Galilei, Descartes, Newton, and Bacon.

With the principle of scientific experiment, Galileo Galilei established the separation of the observer from the observed phenomenon; Rene Descartes formulated the philosophical foundation for this by dividing the world into the thinking substance man (res cogitans) and the external world functioning according to mechanical laws (res extensa); Isaac Newton succeeded in combining philosophical atomism with experimental method; and Francis Bacon, finally, formulated the principle of a technocratically organized scientific community.

According to the Cartesian world view, the conventional scientific approach attempts to isolate a phenomenon and divide it into increasingly smaller units, which are to be studied and described in isolation until, ultimately, the elementary building bricks (atoms) can be identified. A phenomenon can then be reduced to, or explained with reference to, these building bricks. This is often termed a reductionist approach.

Though computer science is a fairly new discipline, it is predominantly based on the Cartesian world view. As Edsgar W. Dijkstra has pointed out: "A scientific discipline emerges with the—usually rather slow!—discovery of which aspects can be meaningfully 'studied in isolation for the sake of their own consistency'" (Dijkstra, 1982).

Reductionist thinking lies at the core of a great many problems. This is because scientists and engineers tend to focus only on those aspects which can be treated in a technical/technocratic manner, neglecting the social and political dimensions of a problem. Yet, despite the enormous progress in computer science since World War II, many computer professionals have become aware of the negative impact of computers on society: terms such as job killer and data protection have become part of our everyday language. Ultimately, the automation of war and the creation of a new golem by means of artificial intelligence would appear to mark the end of the dominance of the human race.

The dehumanization of work has also been seen as an indication that computing technology inevitably leads to a technocratic society unless efforts are made to fight for the democratization of work. "Computers Dividing Man and Work" is the title of a booklet containing descriptions of several Scandinavian research projects, in which scientists, trade union representatives, shop stewards, and employees attempted to work together cooperatively to determine how workplaces could be designed to meet the interests of the users (Sandberg, 1979).

The division of labor is not, however, a phenomenon exclusively related to the invention of the computer. It has to be considered in the wider context of political, cultural, economic, social, and philosophical questions. While for centuries the scientist was concerned mainly with the technical problem of dominating man and nature, today the significance of the social dimension of mastering science and technology has been recognized.

Organizations like CPSR, Science Shops, or the "Forum Informatiker für Frieden und gesellschaftliche Verantwortung (FIFF) e.V.", (comparable to CPSR), are necessary steps along the way, but cannot be our ultimate goal. This would again lead to differing political and scientific cultures, the one dealing with the invention of a new technology, and the other with the problems caused by this technology. Hence, an ecological perspective has to be incorporated into our daily work if we wish to avoid finding ourselves, as computer professionals and CPSR or FIFF members, in the same dichotomy experienced by those working for the Science Shop.

ORIENTATION TOWARDS ECOLOGY

The word *ecology* is derived from the Greek *oikos* (house). According to Webster's dictionary, it is: "1. (a) the branch of biology that deals with the relations between living organisms and their environment (b) the complex relations between a specific organism and its environment 2. Sociology the study of the relationship and adjustment of human groups to their geographical environment" (Webster, 1974, p.442).

Basic ecological ideas can be traced back to the natural philosopher Alexander von Humboldt. In 1827, he gave a series of lectures at the University of Berlin which were published in five volumes entitled "Kosmos. Entwurf einer physischen Weltbeschreibung" (Cosmos. Draft of a Physical Description of the World). Humboldt's basic ideas included: the diversity of natural phenomena as an expression of the life-force; man's affinity with nature; the connection between knowledge and sentience; and the recognition that all living systems are only accessible to experience and comprehension in terms of their history and their embedment in time and space.

It was only through large-scale destruction of the natural environment and the psychological estrangement of man in a highly technicized world that these ideas recaptured man's attention, finding an echo in recent ecological ideas and approaches.

Ecological thinking focuses on the relations between an entity and its environment, instead of analyzing the entity itself. Such relations can only be analyzed by studying the processes through which they are established. Hence, we have to study the processes by which intelligence/intelligent behaviour is revealed to understand the crucial difference between man and machine.

In my doctoral thesis (Keil-Slawik, 1985), I have identified basic concepts common to biological, psychological, and sociological processes in order to attain a conceptual framework for an ecological approach. This can be briefly summarized as follows.

In living systems, the gestalt-forming processes are always based on the interaction of a conservative and a dissipative principle (see Eigen & Winkler,

1983). Generally, the conservative principle may be identified as form, and the dissipative principle as process. The form shapes the process, but does not determine the outcome. If the outcome were determined, no innovative or creative act could take place. On the other hand, if the form were not to shape the process, no gestalt could ever emerge.

A machine, in contrast, is a device designed, built, and used by man for a certain purpose. We want this device to behave in a well-defined, predictable manner, in conformity with our purpose. If an error occurs, the machine will either stop or deliver an incorrect result. In both cases, the designers will sit down and analyze what has gone wrong, either in the design, development, or application, in order to come up with an improved version of the machine or a new one. This conforms to the general notion of design as given by Alexander (1964).

According to Ryle (1949), human errors are "exercises of competence" which serve to improve our skills and enable us to learn. Juergen Habermas, referring to Ludwig Wittgenstein's assertion that it is impossible to obey a rule in isolation, concludes "that the identity of meanings can be traced back to the ability to comply with, together with at least one other subject, rules of intersubjective validity; here, both subjects must possess the competence to behave in accordance with rules and to assess this behaviour critically" (Habermas 1982, p. 34; translated by the author).

Machines lack this competence. If the development and improvement of machines are viewed as an evolutionary process, then machines represent form (i.e., they are passive), whereas man's activities correspond to process (i.e., he plays an active role). According to this view, machines incorporate the knowledge and experience of human beings, embodying the result of intelligent human behavior. They do not behave intelligently themselves. As Michael Polanyi concludes in a more general sense: "no amount of subsequent experience can justify us in accepting as identical two things known from the start to be different in their nature" (Polanyi, 1983).

Ecological thinking focusing on relations is holistic in the sense that the nature of a relationship cannot be expressed by means of attributes or qualities of the entities being related to each other. Thus, the ecological approach views software as a means of promoting communication and learning processes in a complex social and technical environment.

OPEN AND CLOSED SYSTEMS IN SOFTWARE DEVELOPMENT

The notion of open/closed systems is used here to denote different views of software. It serves to illustrate the limitations of the reductionist approach and the possibilities offered by the ecological perspective.

William A. Wulf has given a typical example of a closed system view of software development: "The galling thing about the generally poor quality of much current software is that there is no extrinsic reason for it; perfection is, in principle, possible. Unlike physical devices: (1) There are no natural laws limiting the tolerance to which a program can be manufactured; it can be built exactly as specified. (2) There is no Heisenberg uncertainty principle operative; once built, a program will behave exactly as prescribed. And (3) there is no friction or wear; the correctness and performance of a program will not decay with time" (Wulf, 1979).

This statement is based predominantly on a view of software as an autonomous product. A lot of research is being done to develop mathematical formalisms in order that, once a complete formal specification is arrived at, it may be possible to check mechanically if the document is complete, consistent, and unambiguous. The crucial question is, however, whether such a formalism does actually provide an adequate basis for communication among developers themselves, and between developers and users, during the development process.

In fact, the result of the software development process is a product which is reduced to formal aspects. However, the meaning of such a product for the people involved cannot be defined by mathematical descriptions. Essential for understanding is participation in the reduction process. To understand, for instance, a mathematical abstraction, one has to know the design alternatives underlying the abstraction.

The closed system view assumes that once the topmost functions are formally specified, the whole product can be constructed by specifying the semantics of each function or operation by subfunctions or lower-level operations. This is repeated until, eventually, each (sub)function or operation is defined by elementary functions or operations, which may be regarded as semantic atoms. Such semantic atoms may be machine-executable operations, such as are provided by a programming language and a compiler; or they may be basic mathematical objects, when using a mathematical specification language. In this view of software as a closed system, it is not feasible to introduce new operations whose semantics are not derived from other existing operations or from the semantic atoms. On the other hand, if the topmost functions are specified, the production process consists in dividing functions into subfunctions, whereby at each level "implementation details" are added. During the whole process, though, the identity of the system is maintained by the fact that the same functions remain to be realized.

The conventional approach to software development embodies such a closed system view in two respects. First, the phase model advocated by Boehm (1976) generally assumes that requirements can be determined and fixed in advance, and that the latter phases are merely steps in which the initial document, the requirements specification, is successively transformed into a functioning system, which, it is hoped, fulfils the specified requirements. Second, mathematical

specification techniques, which may be employed in one phase, aim at formally defining the semantics of the system functions in a top-down manner, as described in the last paragraph.

To distinguish the notion of closed system as used here from other definitions, such as, for example, that of Maturana and Varela (1980) and Bertalanffy (1984), the term *productionally closed* is used. This notion emphasizes that closedness refers to the assumption that there is a complete specification of the system to be built, and that the whole production process can be carried out only with reference to the specification, which is regarded to be the theory of the program (see Turski, 1985).

The problem inherent in the closed system view is twofold: no explicit relationship is established between the software and the usage context, that is, the application area, and the development process is characterized purely by attributes which are related to the result of the process, that is, the notion of top-down or bottom-up development, for instance.

According to Jones (1979)—and this is corroborated by our own experience—there is no software system in practical use which has been successfully developed purely on the basis of a written specification. A closer look at the development process leads us to call for an open system view, taking into account the fact that human beings and the software under development together form the system that "maintains itself in a continuous inflow and outflow, a building up and breaking down of components" (Bertalanffy 1969, p. 39).

Software development is part of an organizational process in the course of which production and decision processes are changed, the working routines and communication processes modified, and the organizational structure possibly redefined. These processes of organizational change are imposed by differing interests and views put forward by the people involved. Furthermore, people's behavior and their requirements may have changed or will change by the time the system is installed. New requirements emerge, and experience gained in using the system results in new insights and demands (see Belady & Lehman, 1979; Lehman, 1980).

The same holds for the technical production of software. Software and the related documents are modified during the development process; errors are found; some parts may be optimized; design decisions are revised; and the system, or some part of it, may be restructured. The reasons for, and the motivations leading to, such changes are not usually documented (see, by way of an exception, Parnas & Clements, 1985), and cannot be documented in their entirety and with all their relations and mutual dependencies. Thus, in Naur (1985) programming is viewed as theory building about how the problems in hand are solved by program execution. Each document or product reflects the history and intermediate development stages only to a very limited extent, or, as Peter Naur puts it, "reestablishing the theory of a program merely from the documentation is strictly impossible" (Naur, 1985, p.258).

If we accept this view, it follows that any document or any piece of software can only be understood with reference to the knowledge of those who have participated in the development process and have experience in using the system. Software and the related documents are not autonomous entities, but are part of continuous processes of social interaction, such as communication, learning, understanding, and adapting to changing needs. In the development process, documents and programs serve as the explicitly formulated common memory of the people involved in that process.

The identity of a software system, viewed as an open system, is maintained by the people involved in the development process. Regarding software as an open system means using documents and programs to promote understanding between users and developers; establishing relations between software functions and the usage context; initiating a sequence of cycles of (re)design, (re)implementation and (re)evaluation; in short, providing a flexible framework to cope with changing requirements and the actual needs of the users in an evolutionary development setting.

To give the reader a taste of how this can be approached in practical terms, let me make a brief excursus on our methodological framework STEPS (Software Technology for Evolutionary Participative Systems Development):

> In our research group at the Technical University of Berlin, headed by Professor Christiane Floyd, we have focused our attention on the processes of developing and using software. Basing our approach on a process-oriented view of software development (Floyd, 1981), we began to consider the usage context of a DP system, and the communication and learning processes between developers and users as well as within the development team. Subsequently, we developed a cyclic project model in which we made a clear distinction between the requirements definition—describing what the user wants—and the functional specification—describing what the developer offers as a solution (Floyd & Keil, 1983).
>
> Adequacy of the DP system with respect to the user's work tasks has become one of our primary quality criteria. We have therefore developed a technique for Task-Oriented Requirements Analysis (TORA), which allows us to model working routines and the embedment of DP functions in work tasks (Keil-Slawik, 1987). The models elaborated by the developer serve as the "universe of discourse" when talking with the users, not the system functionality itself.
>
> We work with gestalt-forming project techniques in which we try to distribute knowledge between the various teams involved in the development process and to establish a team spirit, that is, to work cooperatively towards a common goal, and not against each other.
>
> STEPS also includes component methods for dialogue specification and modular design. Depending on the actual problem setting, these techniques and component methods have to be arranged, modified, extended, or even replaced to a certain extent by other techniques and methods, and have to be embedded in an overall

development strategy which includes various ways of prototyping. As might be concluded, STEPS does not aim to replace completely conventional software development methods and techniques. Our aim is to enrich the repertoire of software engineering with a view to human-centred systems development. The general philosophy underlying STEPS is outlined in Floyd (1987).

And what holds for STEPS, applies, in my opinion, in a general sense: I do not claim here that the conventional scientific approach, which has been characterized by the notion of the closed system, lacks validity as a means of solving specific problems. My criticism is leveled at the attempt to characterize human beings and their activities by notions derived from the analysis of computing machinery and their algorithms; and against the use of formal mechanisms for prescribing how people are to react in a given situation.

The closed system view, by neglecting the difference between form and process, invariably leads to machine-centred quality criteria which fail to do justice to human capabilities. William A. Wulf concludes the statement cited at the beginning of this section with the words: "It is only our human frailties that limit the quality of software" (Wulf, 1979).

In a cultural environment in which machines are generally regarded as working more accurately and being less prone to error than human beings (see Michie & Johnston, 1985), the power of a machine or a mathematical formalism is rated higher than the power of human intuition, creativity, and flexibility. Moreover, in keeping with the conventional approach, it is often concluded that we must increase computing power or the powerfulness and rigorousness of formalisms in order to overcome human insufficiencies. In the last instance, it is often argued, with respect to artificial intelligence, that once machines have acquired intelligence they will exhibit all the advantages of human intelligence without its limitations and frailties.

This widely accepted view, at least among authors of popular literature on artificial intelligence (see Keil-Slawik, 1987b), is not backed up by empirical evidence, but is rather an expression of the hopes of machine designers. The more complex a DP system is, the more indispensable the human operators become with respect to reliability. In Celko, Davis, and Mitchell (1983) and Naur (1982), it is demonstrated that the use of machines and formalisms which fail to appeal to our intuition and which force us to think or react in a prescribed manner will invariably produce less satisfactory results than if they were designed to meet human needs. To put it another way: humans make more errors than machines if they have to perform machine-like operations.

RESPONSIBLE SYSTEMS DEVELOPMENT

My arguments in favor of the ecological approach have stressed only one dimension: the relationship between human activities and the tools and techniques being used. More dimensions have to be brought into the discussion.

Let me attempt to illustrate the ecological view with reference to another dimension, namely: values.

We are used to introducing and handling abstract concepts as if they were real objects of our daily experience. During my student days—and even later on—discussions on responsibility ran along lines which I should like to term kitchen knife ethics. This principle denotes arguments such as: "A knife can be used for cutting bread, or to kill someone; it is neutral with respect to its use." The same argument is often applied to computers: They can be used for good or for evil purposes. If we were to accept this argument, there would be no necessity for abolishing the strict separation of scientific and political activities.

My arguments against this conclusion run along two lines:

1. The computer, as such, does not exist.
2. We invariably act in a social context.

Regarding the first, if we look at a single, individual computer, we cannot make any general assertion about it; we can only state: this computer *is* or *is not* . . . Making general assertions implies that we have experience in dealing with a considerable number of computers over a period of time. In studying the history of the computer, I found that its technology had often been shaped by military demands, which were to some extent at odds with scientific or business needs (see Keil, 1982; Keil-Slawik, 1985b). This phenomenon can be encountered in all branches of computer science. It has been demonstrated in the case of Ada by Hoare (1981), and in the case of SDI by Parnas (1985).

Thus, there is a difference between designing a kitchen knife and a bayonet; and between developing systems for business administration, and robots for automating the nuclear battlefield. The general argument makes no sense, since we can neither state that all computers were developed for military needs, nor that none were. Responsibility implies that we are able to tell the difference between a bayonet and a kitchen knife; and means deciding whether we wish chiefly to produce bayonets or kitchen knives. As long as new computers are built, these questions have to be posed anew. The relationship between computer science and, in this case, the military is not a computer-immanent one.

Regarding the second, whenever we tackle a new problem, or have to develop a new system, we rely to a certain extent on our faith in its ultimate success. We expect the system to behave according to the specified requirements, and we believe that we can achieve this by means of the tools and techniques available. It makes a great difference whether we regard the user of a DP system as the weak component that should merely act in a simple and predefined manner, complex decisions being delegated to the computer; or whether we recognize that complex systems are trustworthy (if at all) only because people act freely on the basis of their own experience.

When analyzing the Semi-Automatic Ground Environment air defense system (SAGE), for example, Paul Bracken comes to the following conclusion: "Given the complexity of air defense it is not surprising that informal understandings would evolve to fix problems unanticipated by the system's planners. Very few complex systems would ever run if rule books were followed to the letter" (Bracken, 1983, p. 12). These oral agreements between operators never showed up in official reports. Even though SAGE didn't work (see Fallows, 1981), increasingly more complex systems such as the World Wide Military Command and Control System (WWMCCS) have been designed and built without any reliable demonstration that the problems which had plagued the designers of SAGE and similar systems had been adequately solved (see Keil-Slawik, 1986). With the proposal of a battle management system for SDI, it is argued that most of the complex decisions involved in battle management have to be performed by computers owing to time and complexity constraints.

The main argument put forward by David Parnas when he resigned from the SDIO panel was that the intended computerized battle management system would not be trustworthy. Again, it should be noted that trustworthiness denotes a relation between humans and technology, and is not in itself an attribute of the computer. It is based on our previous experience in the development und use of complex DP systems and thus cannot be replaced by "in principle" arguments based on certain newly developed techniques or tools. Thus, the question Parnas tried to answer was: Has there ever been a complex software system which has been successfully built and reliably operated under circumstances and constraints which are comparable to those formulated in the SDI report (Fletcher, 1984)? The answer is: Definitely not!

Responsibility implies that we have to make an individual decision in any such case. Christiane Floyd defines three types of limits for the responsible use of computers (see Floyd, 1985):

- the technical limits, i.e., where computers are utilized as a result of misguided trust in the capabilities of computer programs
- the human limits, i.e., where computers are used as a result of the misguided equating of people with machines
- the ethical and political limits, i.e., where the attempt is made to do things with computers that ought not be done with them

These limits should be regarded as a first step towards formulating principles which are generally accepted within the computer science community.

To sum up, it may be stated in general that the ecological perspective compels us to make a clear distinction between what can be said about, and concluded from, the properties and structure of form (be it a product, a document, a formalism, a tool, etc.), and what can be said about, and concluded from, the process, that is, the genesis of form as denoted, for instance, by evolution,

growth, learning, thinking, and designing. These processes can only be characterized by the relations established between the individual person or entity and the environment.

We have to make sure that no confusion arises between the result and the process of producing it. This applies to the use of formal techniques in software development, in particular, as well as to the realm of artificial intelligence in general. If we equate humans with machines, we confuse the result of intelligent human behaviour (the machine) with the process which produced it. Since we can never deduce from the final structure the laws governing its genesis and control, we should never use DP systems or formal techniques to prescribe how people are to behave. When designing products, we should ensure that they exhibit a high degree of flexibility so as to meet the actual needs and intentions of the users.

To show how this can be achieved, I will refer to the notion of "small systems" as adopted by Dirk Siefkes. He attempts to provide guidelines for creating and maintaining systems appropriate to human needs. As he points out, only small systems evolve. A small system, according to Siefkes' definition, requires the participation of human beings. He defines a system as being a group of people united by a given task, and supported or hindered in this task by natural or technical means. In a small system, six dimensions have to be continually balanced: means, rules, words, concepts, values, and will. These dimensions Siefkes calls *bearings*. Rules may be rigid or uncertain; knowledge may be fixed or shaky and so on. "A system is small if it is appropriate, i.e. neither excessive nor defective, in all bearings" (Siefkes, 1987, p. 3).

Since these bearings are constantly changing, they have to be continually rebalanced with respect to the actual processes, establishing the relationship between means and value, rules and will. Thus, if we separate development from use, or the production of scientific results from their application, we cannot hope to design small systems. What is more, we cannot talk about responsible systems development, while neglecting the link between our daily work as computer professionals and the social context in which it is embedded.

CONCLUSION

In the previous sections I have tried to give a brief outline of what may be termed ecological thinking, contrasting it with the conventional reductionist approach. As may have become apparrent, a lot of questions are left open: What relationships should be considered relevant? What are the links between the various relations? How do we determine where the conventional approach may be sufficient? How can we develop operative and constructive principles? And so on.

Reductionism underlies every act of articulate communication. Thus, the ecological approach presented here is also reductionistic, particularly when

transformed into rigorous scientific method in the conventional sense. My main argument in favor of the ecological approach is that ecological thinking embodies a world view which makes us aware of our being embedded in an evolving universe and connected to our environment through our own history as living beings. There exists a dialectic relationship between our acting as autonomous beings upon the world, and at the same time the world's acting upon us. A fundamental conclusion which may be drawn from this fact is that we can only ever control the world to a very limited extent. This discovery may engender in us a feeling of deep respect for the living world, which we can only totally subjugate by destroying it.

The ecological approach attempts to unite things which have been separated in the reductionist approach: science and policy making; facts and values; development and use; mind and body; subject and object; theory and practice; and so on. However, in uniting science and policymaking we have to ensure that we arrive at a sound intersubjectively acknowledged basis for conducting our work. This represents a completely new problem for computer professionals and engineers, and we have no deeply rooted experience to build upon. On the other hand, there are strong indications that the same basic principles may apply in biology, sociology, and especially in psychology (see Neisser, 1976; Gibson 1979). This may pave the way for future interdisciplinary work; software development, in particular, needs an interdisciplinary approach of this sort.

Gregory Bateson has, in my opinion, formulated the most fundamental insights into the questions addressed here. He states: "Learning the contexts of life is a matter that has to be discussed, not internally, but as a matter of the external relationship between two creatures." And he concludes: "Relationship is not internal to the single person" (Bateson, 1980). And I would add: Nor to a group or subculture.

As Bookchin (1982) has also pointed out, the ecological perspective should not be limited to science, or even to organizations like FIFF or CPSR. The fact is that ecological subcultures can only exist within an ecological society.

REFERENCES

Alexander, C. (1964). *Notes on the synthesis of form*. Cambridge, MA: Harvard University Press.

Bateson, G. (1980). *Mind and nature: A necessary unity*. New York: Dutton.

Belady, L.A., & Lehman, M.M. (1979). The characteristics of large systems. In P. Wegner (Ed.), *Research directions in software technology*. Cambridge, MA: MIT Press.

Berman, M. (1981). *The reenchantment of the World*. Ithaca, NY: Cornell University Press.

Bernal, J.D. (1965). *Science in History* (Vol. 1–4). New York: Hawthorn Books.

Bertalanffy, L.V. (1984). *General system theory.* New York: G. Braziller.

Beuschel, W., Bickenbach, J., & Keil, R. (1984, March). Informationstechnologie—technische Moeglichkeit und taegliche Nutzung. *7th International Conference on Data Processing in Europe,* (pp. 19–23). ADV Arbeitsgemeinschaft fuer Datenverarbeitung, Wien.

Beuschel, W., Bickenbach, J., & Keil, R. (Eds.). (1983). *Computer* in Alternativprojekten. *WILAB-Bericht 1-83.* Berlin: Wissenschaftsladen Berlin.

Boehm, B.W. (1976). Software engineering. *IEEE Transactions on Computers, C-25*(12), 1226–1241.

Bookchin, M. (1982). *The ecology of freedom.* Palo Alto, CA: Cheshire Books.

Bracken, P. (1983). *The command and control of nuclear forces.* New Haven, CT: Yale University Press.

Celko, J., Davis, J.S., & Mitchell, J. (1983). A demonstration of three requirements language systems. *SIGPLAN Notices, 18*(1), 9–14.

Dijkstra, E.W.D. (1982). *Selected writings on computing: A personal perspective.* New York: Springer-Verlag.

Eigen, M., & Winkler, R. (1983). Das Spiel. Naturgesetze steuern den Zufall. Mmuenchen Zuerich. Berlin: Springer.

Fallows, J. (1981). *National Defense.* New York: Random House.

Fletcher, J. (1984, February). Report of the study on eliminating the threat posed by nuclear ballistic missiles (Vol. V): Battle management communications, and data processing (Contract MDA 903 84 C 0031; Task T-3-191).

Floyd, C. (1987). Outline of a paradigm change in software engineering. In G. Bjerknes, P. Ehn, & M. Kyng (Eds.), *Computers and democracy. A Scandinavian challenge.* Brookfield, VT: Gower.

Floyd, C. (1985, Spring and Summer). Responsible use of computers: Where do we draw the line. *CPSR Newsletter, 3*(3 and 5), 2–3, and 1,2.

Floyd, C. (1981). A process-oriented approach to software development. *Systems architecture: Proceedings of the 6th European Regional Conference.* Guildford, England: Westbury House.

Floyd, C., & Keil, R. (1983). Adapting software development for systems design with users. In U. Briefs, C. Ciborra, & L. Schneider (Eds.), *Systems design for, with, and by the users.* Amsterdam: North Holland.

Gibson, J.J. (1979). *The ecological approach to visual perception.* Boston: Houghton-Miflin.

Guralick, D.B. (1982). *Webster's New Word Dictionary of the American Language.* New York: Simon & Schuster.

Habermas, J. (1982). Theorie des kommunikativen Handelns (Vol. 1–2). Frankfurt: Suhrkamp.

Jones, C. (1979). A survey of programming design and specification techniques. In *Proceedings of the Conference on Specifications of Reliable software.* New York: IEEE Computer Society.

Hoare, C.A.R. (1981). The emperor's old clothes. *Communications of the ACM, 24*(2), 75–83.

Keil, R. (1982). Die neue Waffe—der computer. In H. Nehmer (Ed.), *GI—12. Jahrestagung.* Berlin: Springer.

Keil-Slawik, R. (1988). Von der mechanisierung des Kopfes zur Oekologie des Geistes. Ein Literaturueberblick zu anthropologischen Aspekten von menschlicher und kuenstlicher Intelligenz. In M. Stoehr & H. Wendt (Eds.), *Menschliche und Kuenstliche Intelligenz*. Frankfurt: Fischer.

Keil-Slawik, R. (1987, May 12–15). Supporting participative systems development by task-oriented requirements analysis. *Proceedings of the International Federation of Information Processing Societies. Working Group 9.1 Conference on System Design for Human Development and Productivity: Participation and Beyond*. Berlin, GDR.

Keil-Slawik, R. (1986). SDI considered harmful—ansaetze zum umdenken in der softwaretechnik. In A. Shulz (Ed.), *Die zukunft der informationssysteme*. Berlin: Springer.

Keil-Slawik, R. (1985a). KOSMOS—Ein konstruktionsschema zur modellierung offener systeme als hilfsmittel fuer eine oekologisch orientierte softwaretechnik. Dissertation, Technische Universität, Berlin.

Keil-Slawik, R. (1985b). Von der Feuertafel zum kampfroboter—die entwicklungsgeschichte des computers. In J. Bickenbach, R. Keil-Slawik, M. Loewe, & R. Wilhelm (Eds.), *Militarisierte informatik. Forum informatiker fuer Frieden und gesellschaftliche*. Berlin: FIFF Berlin.

Lehman, M.M. (1980, September). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE, 9*, 1060–1076.

Maturana, H.R., & Varela, F.J. (1980). *Autopoiesis and cognition*. Boston: D. Reidel.

Michie, D., & Johnston, R. (1984). *The creative computer*. New York: Viking.

Naur, P. (1982). Formalization in program development. *BIT, 22*, 437–453.

Naur, P. (1985). Programming as theory building. *Microprocessing and Microprogramming, 15*, 253–261.

Neisser, U. (1976). *Cognition and reality*. San Francisco: W.H. Freeman.

Parnas, D.L. (1985). Software aspects of strategic defense systems. *American Scientist, 73*, 432–440.

Parnas, D.L., & Clements, P.C. (1985). A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering, SE-12*(2), 251–257.

Polanyi, M. (1962). *Personal knowledge. Towards a postcritical philosophy*. Chicago: University of Chicago Press.

Ryle, G. (1949). *The concept of mind*. New York: Barnes and Noble.

Sandberg, A. (1979). *Computers dividing man and work*. Stockholm: Arbetslivscentrum.

Turski, W.M. (1985). *Informatics. A propaedeutic view*. Amsterdam: North Holland.

Wulf, W.A. (1979). Introduction to part I: Comments on "current practice." In P. Wegner (Ed.), *Research directions in software technology*. Cambridge, MA: MIT Press.