Aesthetics and the Human Factor in Programming

A.P. Ershov

The Luncheon Address at the 1972 Spring Joint Computer Conference was given by Professor Andrei P. Ershov, Information Division Head of the Computer Center, Siberian Division, USSR Academy of Sciences, Novosibirsk University. Many who attended felt his remarks were particularly pertinent and thoughtful. Communications is grateful to the author and to AFIPS and the SJCC Technical Committee for the opportunity to present his speech to a wider audience.

I must admit that when I received the invitation to appear at this luncheon, my first impulse was to find out who before me had been honored with a similar invitation. I found in a list of preceding banquet and luncheon speakers three famous writers, two Congressmen, one general—but hardly a representative of our own computer profession. Probably that was not accidental: the rapid and extensive development of our industry has inevitably increased the number of its peripheral contacts, the consequence of which has been a growing interest in the outside world. However, it is obvious that in order to cope with the serious problems of our profession, we must not only look at ourselves in the mirror of public opinion, but must also probe inside ourselves. I shall therefore allow myself the liberty of bringing to your attention some observations concerning the human factors vital in programming.

I begin with the following remark: the present is a time of difficulty for programmers. The volume of work to be done is increasing; wages less so. The romantic aura surrounding this inscrutable occupation is, if it ever really existed, beginning to fade. Software houses are melting like snow in spring. Professionals accustomed to being strongly in demand now find themselves waiting on the

books of the employment agencies. Even the claim of programmers to be a special breed of professional employee has come to be disputed. Still more significant, authority over the freewheeling brotherhood of programmers is slipping into the paws of administrators and managers—who try to make the work of the programmers planned, measurable, uniform, and faceless.

I do not mean to deplore this tendency. I myself have for a long time been related to programmers as an administrator and, as such, have contributed by providing background support to the systematic organization of their work. As an administrator, I may remark that the network of relationships connecting programming to the world at large has not yet reached maturity. Consider for example the proliferation of software houses during the last decade. A small software house, which at its smallest might better be called a software hut, will often be nailed together in the course of several weeks by a group of able programmers, who may perhaps plan on leaving a large organization in which they have received their initial experience. What motivates such an initiative? In many cases, it is a mix of profit-thrust and a somewhat unrealistic desire to break free of excessive supervision, accompanied of course by some particular interesting and useful technical software idea. However, only those groups have survived in which initial motives were quickly subordinated to tough economies, and in which hierarchical interpersonal relations and strong discipline were imposed—in other words, all of those things reconstructed which originally will have driven a software group from its parental roof. The history of such enterprises can at times be reminiscent of the adventures of the three little pigs: the brotherhood of programmers finally winds up in a well-fortified software house, but not until a fraction of them are swept away by the wolfish wind of merciless commercial pressure. Do not think here that I refer exclusively to American reality; I have more direct personal experience in this connection.

Thus the subordination of programming to big enterprise is an unavoidable fact. However, I see a certain danger in converting programmers into what is simply a highly paid subgroup of the working class. If such a tendency is to be resisted, a programmer must find some system of inner values in his specialty, values which can help him both to assimilate industrial working methods and, when necessary, to transcend them.

This last remark brings me to my second theme: that such a system of values is objectively inherent in programming. Since, however, these values are not fully conscious, they require explanation and defense. The value system to which I allude has many bases, the most material of which, perhaps, is the professional status of the programmer (and I speak here of programmers in a broad sense, including systems analysts). I shall however in this address emphasize not material issues, but shall concentrate on the aesthetic or emotional aspect of programming; and discuss not the compensation which comes to a programmer when he goes with his products out into the marketplace, but the moral forces which affect him when he is left alone with his program and his computer.

You may, of course, ask me: is it worth discussing this nonmaterial aspect of the situation? I answer that it is, and only in part because programming has become a massive human activity.

In my opinion programming is, for at least the following three reasons, the most humanly difficult of all professions involving numbers of men:

1. Programmers constitute the first large group of men whose work brings them to those limits of human knowledge which are marked by algorithmically unsolvable problems and which touch upon deeply secret aspects of the human brain.
2. A programmer's personal push-down stack must exceed the depth of 5–6 positions, which psychologists have discovered to characterize the average man; his stack must be as deep as is needed for the problem which faces him, plus at least 2–3 positions deeper.
3. In his work, the programmer is challenged to combine, with the ability of a first-class mathematician to deal in logical abstractions, a more practical, a more Edisonian talent, enabling him to build useful engines out of zeros and ones, alone. He must join the accuracy of a bank clerk with the acumen of a scout, and to these add the powers of fantasy of an author of detective stories and the sober practicality of a businessman. To top all this off, he must have a taste for collective work and a feeling for the corporate interests of his employer.

I know that to overcome these difficulties, great emotional effort is required, which can be provided only through conscious deliberation and a positive inner attitude. If an Arthur Haley wrote a novel, "The Computer Center," it would surely rank as the most engrossing of bestsellers!

Which brings me to my next thesis. An understanding, a feeling for the aesthetic of programming, is needed, and not only as a driving force for the programmer: it is necessary for those who manage programmers, and especially for those who educate and train them. Let me cite some of the hard organizational dilemmas whose solution can only be reached if we take account of all the

aesthetic factors which have been noted:

- Is it necessary, is it possible to organize software development using an assembly-line approach?
- Is it possible or necessary to separate the design of large programs from their implementation?
- In organizing software projects, which are more difficult to find (and why)—managers or performers?
- How can the elite and the mass character of systems programming be harmonized?
- What individual capabilities should we look for in the programmer? Are these qualities specific enough that we can take them into administrative account?
- How should programmers be trained? By cultivating a "world view" (in universities) or by emphasizing "job-related" skills (in technical institutes)?

All these questions are only part of the general problem of software management. I shall of course not ruin your digestion by attempting a systematic analysis of this very convoluted problem. Let me undertake however to make a few personal comments in an attempt to relate these organizational problems to an analysis of the human factor in programming.

First a comment concerning assembly line methods in software development. I personally consider the assembly line a most diabolical invention. True, it raised productivity to an unheard of level. At the same time, however, it gave the labor situation a faceless character to an extent not previously known. In programming, the use of assembly-line organization can destroy the intellectual work-satisfactions which motivate programmers, and the contradiction between the monotony of work so organized and the difficulty of a programmer's work can bring about neurosis. Imagine a man who is compelled to work exclusively at solving crossword puzzles, eight hours a day, five days a week, 50 weeks a year, and you will understand what a programmer specialized in the writing of some narrow class of program components faces. Surely, therefore, to increase programmer productivity by the division-of-labor principle, i.e. by breaking down the programmer's task into elementary operations, is far from a simple matter.

Concerning managers and implementers: a manager's ability to deal with a technically specific work force is often limited. Often, a project manager will prefer to assign implementation responsibility to a young specialist just two or three years out of the university, rather than to a person with much more substantial experience. Does this not indicate a preference for the easy plasticity of a young man, an attempt to avoid grappling with a more mature and resistant 33-year-old family head? Does it not also show that we don't know how to enhance the professional dignity of an implementer; how to help him avoid losing ground with age; how to keep him useful, not only for the manager with whom he must work, but for himself and for his future managers?

Concerning "world view" versus "job skills," the problem is of course not only that of estimating the optimal ratio between Ph.D.s and software technicians, though this problem raises staffing questions which have faced many managers—the essence of the problem is that programming requires of a man a certain attitude, a certain moral preparation for his duty. The programmer is the lynchpin of a second industrial revolution; as such he must possess a revolutionary way of thinking.

That programmers are an elite group is quite evident to me. In this respect the activity of programmers represents a significant challenge to humanity as a whole; a challenge which I hope will be accepted. I will return to this thought below.

I now want to restate the central thesis of my speech and affirm that programming embodies rich, deep, and novel aesthetic principles on which the inner relationship of a programmer to his profession is based, those principles which give him both intellectual and vivid emotional satisfaction. This aesthetic has roots in the creative nature of programming, in the difficulties which programming overcomes, and in the social significance of programming.

To define the core aesthetic of any professional activity is not a simple task. Such an aesthetic is realized by a set of subjective attitudes which link professionals together. I quote the Russian proverb: "One fisherman recognizes another fisherman from afar." A professional aesthetic influences and is influenced by the ethical code of a profession, by the technical subject matter of the profession, and by the profession's juridical status. The account of the aesthetic components of programming which I shall now try to give will accordingly also have a subjective and preliminary character.

I begin with some remarks concerning the inner nature of the programming aesthetic.

The creative nature of programming does not require special proof. Indeed, I may assert that, in its creative nature, programming goes a little further than most other professions, and comes close to mathematics and creative writing. In the majority of other professions, even if we put the tiger in the tank, we only tame the forces of nature. We simply use physical and biological phenomena, hopefully in a cleverly economical way, but without understanding

their innermost principles. In programming, however, we go, in a certain sense, to the root. One of the theses of modern epistomology states that "we understand what we are able to program." This phrase vividly characterizes the "maximalism" of our profession.

Another very important aesthetic characteristic of programming is that exceptionally high accuracy is demanded of its finished products. Of course, this is characteristic of much engineering. However, programming requires accuracy going beyond that needed in other types of engineering. There exists a striking contrast between an almost finished and a fully implemented and debugged product. This requirement for 100 percent fulfillment is not only the source of some of the programmer's most frustrating difficulties, but at the same time it gives rise to some of the deepest satisfactions of his work.

Intelligence itself is manifest in the perfected machine-program combination. The programmer plays a full trinity of roles in this familiar miracle. He feels himself to be the father-creator of a program, the son-brother of the machine on which it runs, and the carrier of the spirit which infuses life into the program/machine combination. This triumph of intellect is perhaps the strongest and the most characteristic aspect of programming.

In using a machine, an honest programmer displays one more peculiarity. He relates to it as a good jockey relates to his horse. Knowing fully the possibilities which the machine affords, he will nevertheless not allow himself to indulge in a personal intellectual laziness implying reckless expenditure of computational resources. This essentially aesthetic relationship of the programmer to his work constitutes a most effective safeguard against the mindless accumulation of software inefficiencies, which, though it might not raise any special objections on the part of companies which sell machine time, would cheat the consumer and lose for us the full power of machines.

A second group of aesthetic issues relates programming to its social and public functions. On first meeting and attempting to analyze a social phenomenon of grand scale (and the arrival of the computer on the historical scene is without doubt such a phenomenon) we search for historical analogies broad enough to give us a basis for extrapolation and prognosis. It is in this sense that we speak of the advent of the computer as a second industrial revolution characterized by the industrialization of intellectual work. Another analogy on the same scale may be offered for the profession of programming. The progressive expansion of software is, it seems to me, comparable in many ways to the phenomena set in motion by the invention of printing. The accumulation of books, each one embodying its author's view of the external world, broadened a social process of understanding. In the same way, programs and data banks accumulate informational and operational models of the world, and allow us not only to influence but also to predict the world's evolution, giving us in this way an unheard-of power over nature.

To be a good programmer today is as much a privilege as it was to be a literate man in the sixteenth century. This privilege leads the programmer to expect recognition and respect on the part of society. Unfortunately, such an expectation is not always realized. Let me say that if this recognition is to be granted, work is required on both sides. On his part, the programmer must accept a general ethical principle applying to all professionals, but having a special interpretation for programmers. Three main attitudes to work may be distinguished:

Work for work's sake.
Work for money.
Work for a goal.

In the programmer's world the first two motivations occupy first place, though it is the third that should be most absolute. Therefore I wish to say that the programmer can achieve a fully harmonious relation with society only when loyalty to the goal of which his programming is only part is integrated into his inner attitude.

In speaking of the social functions of programming, I cannot avoid observing that an unsolved technical problem, namely that of giving programming a cumulative effect, stands as an obstruction to the realization of programming's full social impact. To solve this problem is a very complicated task but an absolutely necessary one. The spectrum of opinion about it is boundless. Some say that only a tiny part of presently working programs are of lasting value; others say that OS/360 is already an immortal system of programs. To relate this question to the theme of my talk I want to say that enabling the programmer to see his product as having long-term and stable use will have a decisive effect on his professional self-view.

Allow me now to move on to consider some of the other problems enumerated above. First a few words concerning individual abilities in programming. We hold before us—we need—an image of the ideal programmer. Of course, this is a mythical person, but who said that our profession can do without myths and fairy tales? Every one of us must have at least once seen or heard about a wonder programmer from whose programs not one instruction can be deleted and who writes a thousand

bug-free instructions a day. It is in the nature of man to seek for ideals and examples. Precisely to provide such a reference point I will add a remark to the continuing discussion of "prima donnas" in programming. To declare flatly that they are undesirable is at the very least shortsighted. I have been lucky enough in my life to meet several such prima donnas who, despite their individuality and extravagance, made priceless project contributions, especially in difficult situations. So that I definitely advocate complete recognition and full exploitation of the broad scale of individual abilities in programming.

Let me now go on to discuss the relationship between design and implementation in software. Conflicting forces affect this relationship. Managers responsible for major projects seek ways to formalize the distinction between design and implementation, hoping to allow the task of implementation to be shifted from person to person in an administratively convenient way. On the other hand, the work itself offers terrific resistance to such a division. Let me say that a correct resolution of these conflicting forces is impossible without taking into account vital human factors and the aesthetic needs which they imply (needs which create obstacles to the implementation of passively received ideas). Moreover, to give one's own technical project into strange hands is equivalent to sending one's children to a boarding school. Such action, even if it becomes necessary, is still full of loss.

I have, in developing my argument, characterized programmers as an elite, and have stressed the very special nature of programming and the far-reaching demands it makes upon limited human ability. In winding up I would like to return to this issue and to view it differently. When I was last in the United States, in 1970, I was very much impressed by the new ideas in the education of children developed by Marvin Minsky and Seymour Papert of M.I.T. Minsky and Papert threw overboard the cliché that children learn subconsciously by imitation. They proved that men learn best when they form flow-charts of action in their heads, when subroutines are separated out and informational connections traced. Using the problem of juggling with two balls, and appealing to my abilities as a programmer, Professor Papforces is impossible without taking myself wouldn't be able to learn in several hours, thus converting me to his faith forever. This shows that man can greatly strengthen his intellect, if he is able to integrate into his nature the habit of planning his actions, of working out general rules, and of applying them to concrete situations: to organize rules; to express them in a structured way; in other words, to program.

In past ages, the ability to read and write was considered a rare, God-given gift—the destiny of a limited group of the specially chosen. In the present age of general literacy, we perceive reading to be a universally attainable accomplishment, but we are tempted to single out a new elite group, who become arbiters between the lay generality of mankind and the arcane informational model of the world hidden in the machine. Is it not however the highest aesthetic idea of our profession to make the art of programming public property, and thereby to submerge our elite exclusiveness within a mature mankind?

I must ask the audience to forgive my high-flown style. General literacy was the accomplishment of an historical millenium. To accomplish the further step which I have projected, much less than a millenium will be needed, even though we are far from the projected goal. Of course, far more prosaic and pressing problems surround us. Nevetheless, every programmer, even when he is merely leafing through a manual or fumbling to find the right key on a terminal, contains within himself boundless depths of thoughts, desires, and emotions. I am thoroughly convinced therefore that the colleagues and managers of a programmer must understand his professional motives and perspectives.

I have in my remarks discussed certain current problems connected with the human factors in programming. Perhaps a still more important problem has been neglected. At present we limitlessly recruit young people into programming, promising them the sky. However, human generations change much more slowly than machine generations. We, as innovators, must learn to keep a 50-year-old programmer as useful as a new recruit. In 30 years we'll have a million such programmers. Presently we must admit that we don't even possess an approach to the utilization of veteran programmers under modern conditions of change and instability, that we do not know how to make the profession of programmer lifelong and leave its practitioners with a comfortable feeling of personal security.

I would like to express my gratitude to the organizers of this conference, and especially to Dr. Jack Bertram and to Professor Jacob Schwartz for affording me the honor of appearing at this conference. I thank you.