PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT.

Mental Ergonomics As Basis For New-Generation Computer Systems

M.H. van Emden
Logic Programming Laboratory, Department of Computer Science
University of Victoria, Victoria, B.C., Canada V8W 3P6
vanemden@csr.uvic.ca

Abstract

Reliance on Artificial Intelligence suggests that Fifth Generation Computer Systems were intended as a *substitute for* thought. The more feasible and useful objective of a computer as an *aid to* thought suggests *mental ergonomics* rather than Artificial Intelligence as the basis for new-generation computer systems. This objective, together with considerations of software technology, suggest logic programming as a unifying principle for a computer aid to thought.

1 Introduction

When surveying the literature on computing, it is remarkably difficult to find work directly aimed at making computers usable as a *tool for thought*. Even when we go to publications specialized in Artificial Intelligence, we find mostly work aiming at simulating or automating human intellectual functions, but very little on how to use computers to *augment* the intellect in the way envisaged by pioneers such as Licklider, Engelbart, Taylor, and Kay.

Until recently it was understandable that the goal of augmenting the intellect had to be deferred, and that top priority had to be given to the development of hardware and systems software that provided a functional basis on which to proceed towards the main goal. I believe that this basis now exists and that therefore the top priority in computing should be to use the existing machinery to make computers available as tools for thought. It seems, however, that at present it is still top priority to make computers faster, bigger, and cheaper. This can only be explained as a form of inertia: we feel comfortable in an enterprise blessed with past and continuing success and it is painful to change emphasis, even if it is towards what is now *really* important.

Let us then take stock and see what progress has been made towards providing the basis for the goal of making a computer into a tool for thought. The hardware dreamed of by the pioneers has arrived: fast processors, large memories, sophisticated and comfortable displays, high-performance networks; all of this available in thousands of enterprises and institutions. Still, we do not use computers as tools for thought in the way the pioneers envisaged. Were they unrealistic in their expectations? Or is it the case that the remaining obstacles can be overcome?

I believe that the latter is the case, and that the remaining barrier is *the difficulty of using software*. What is from the larger viewpoint but a tool among many, such as, for example, a database program, is so complex that it comes with a fat manual and a programming language of its own, so that to become an effective user is almost a career in itself.

This is but one example of many, of a wider phenomenon I refer to as *concept fragmentation*: that each job seems to require its own special-purpose solution; even worse, that the same job in a different context requires a different solution. This, not hardware, is now the main barrier standing in the way of using computers to augment the intellect.

In this paper I will argue that the best bet for a unifying principle to overcome this barrier is *logic programming*. As Artificial Intelligence often comes up in discussions about how to make computers easier to use, it is important to distinguish the roles to be played by Artificial Intelligence and Mental Ergonomics[1]. For this reason, I sketch the argument from scratch: why there is reason to believe that computers can be tools for thought, before going on to explain in what way logic programming, as ergonomic principle, can help demolish the main barrier now holding up progress.

2 A synopsis of the argument

This section is what is sometimes called "executive summary." Each paragraph summarizes one of the following sections of the paper.

[1]Webster's Third International Dictionary: ergonomics — an applied science concerned with the characteristics of people that need to be considered in designing and arranging things that they use in order that people and things interact most effectively and safely.

Towards Computer-Aided Thought. Writing is *paper-aided thought*: while we can do simple sums in our head, we need help for more complex ones; help offered traditionally by writing on paper. Similarly, we can do simple thoughts in our head; to work out complex thoughts, such as plans, proposals, essays, reviews, critiques, we need writing.

Computers are now widely used as a more convenient writing tool than a pen on paper. The availability of programs for spreadsheets, databases, and communications provide a tantalizing glimpse of a more powerful aid to thought than the pen on paper ever was. Such a potent new mixture deserves a name. I chose one suggested by the familiar concepts of Computer-Aided Instruction (CAI) and Computer-Aided Design (CAD). I call it Computer-Aided Thought, CAT for short. Today's laptop computers already pack the hardware required to support a powerful CAT system. Thus you will be able to take it wherever you go, like a Sony *Walkman*. Let us call such a package "CATMAN."

And hardware trends suggest that CATMAN will be widely affordable, giving unprecedented power to intellectual workers of all ages: school children as well as professionals, business persons, and scientists.

Why Computer-Aided Thought is an underdeveloped area. In spite of spectacular advances in computing, both in large systems and in personal computers, no one, not even the most privileged researcher, has a computer available as a congenial tool for intellectual work. At best she[2] can call on a hodge-podge of language processors, databases and application packages requiring a bevy of system gurus at her beck and call if she is to avoid devoting her career to mastering the mechanics of the various systems.

Improvement is a matter of ergonomics, not AI. A congenial tool for intellectual work needs to handle a variety of tasks: including database work, text processing, communication, constraint exploration, developing algorithms. This diversity of tasks has caused a proliferation of languages so that logically identical tasks need to be done in an exasperatingly different way, just because they occur in a different application. The problem is one of *ergonomics* (in this case Mental Ergonomics): the lack of a unifying concept makes current program interfaces conceptually fragmented. This is where we should look in the first place for help, rather than to Artificial Intelligence.

Logic programming meets the requirements of Mental Ergonomics. I mention some ergonomic principles that help to make computers easier to use and review results showing how such principles can be implemented by means of logic programming.

[2]or "he." Here, as elsewhere in my writings, no gender is to be inferred when none is implied by the context.

3 Towards Computer-Aided Thought

What is "thought"; what is "intellect"? Why do I consider writing "paper-aided thought"? The analogy about complex thought spilling over to paper, just as complex sums do, is due to Susan Horton [1982], who took as starting point the familiar phenomenon that we don't sit down to write an essay with its main line of reasoning ready in our head. Instead, we only *discover* what we want to say as a result of initially unsuccessful and often frustrating attempts to write down inchoate, preliminary versions. In this way, Horton concludes, writing allows us to have thoughts too hard to do in our head.

Another way of expressing Horton's idea is to say that writing "augments the intellect." In 1963 Douglas C. Engelbart published the paper "A conceptual framework for the augmentation of man's intellect" [Engelbart 1963]. Its first paragraph, which I quote in full, contains a better description of "aids to thought" or "augmentation of intellect" than I can give.

> By "augmenting man's intellect" we mean increasing the capability of a man to approach a complex problem situation, gain comprehension to suit his particular needs, and to derive solutions to problems. Increased capability in this respect is taken to mean a mixture of the following: that comprehension can be gained more quickly; that better comprehension can be gained; that a useful degree of comprehension can be gained where previously the situation was too complex; that solutions can be produced more quickly; that better solutions can be produced; that solutions can be found where previously the human could find none. And by "complex situations" we include the professional problems of diplomats, executives, social scientists, life scientists, physical scientists, attorneys, designers — whether the problem situation exists for twenty minutes or twenty years. We do not speak of isolated clever tricks that help in particular situations. We refer to a way of life in an integrated domain where hunches, cut-and-try, intangibles, and the human "feel for a situation" usefully coexist with powerful concepts, streamlined terminology and notation, sophisticated methods, and high-powered electronic aids.

Conventional computer applications are much further developed in the area of what Engelbart calls "powerful

concepts, streamlined terminology and notation, sophisticated methods." I regard the area of "hunches, cut-and-try, intangibles and 'human feel for the situation' " the one where writing helps as an aid to thought in the sense of Horton. Engelbart's vision will be realized when a computer can be used as a congenial tool for writing, giving fluent access to spreadsheets, databases (local as well as remote), numerical and statistical libraries and so on.

Other prophetic early papers that improved upon much later thinking are by Bush [1945] and Licklider [1960]. For an overview of Engelbart's subsequent work in the Augmented Knowledge Workshop, see [Engelbart 1988].

4 Why Computer-Aided Thought is an underdeveloped area

A present-day personal computer can provide a word processor, a spelling checker and a thesaurus. This combination is a powerful advance over pen and paper, and therefore qualifies as a correspondingly powerful aid to thought. Personal computers can also run packages for databases, spreadsheets, and computer mail. But although these are potentially valuable extensions, the resulting combination is not easy enough to use to qualify as a computer tool for thought. The existence of the components conjures up a tantalizing vision of such a tool, but the reality of ergonomics turns the vision into a mirage.

To appraise the situation, let us consider what personal computers have given us, and what's lacking.

What we do have. The amazing thing about personal computers is that they have caused such large step in the direction of a computer tool for thought. The early computers were operated in a closed shop, to which users submitted their jobs, which were collected in batches and run.

Timesharing brought a dramatic change: from a turnaround time of hours to instantaneous interaction. Effectively, the timesharing user has the machine to himself. And even in the sixties, these machines were not small compared to contemporary personal computers. So it was not obvious that a dramatic change would result from the next step, the introduction of the personal computer. Yet they made an enormous difference and that was because of their low cost. This has two effects:

1. Small firms and individual software designers can afford machines of their own.

2. The potential financial rewards in the software market became much greater.

The result should convince sceptics of the power of a sufficiently free market: it resulted in an unprecedented improvement in user interfaces.

This is the more amazing when we realize that since the early sixties timesharing computers have been used for word processing. These installations commanded the best in programming talent and were largely devoted to research. Yet nothing was produced that can compare with the better word processing packages that appeared on the market soon after personal computers took off. For most of the 1980's, Unix-based workstations, with more powerful hardware than personal computers, had word processors worse than those on personal computers. Spreadsheets are an even more striking example. This type of software, now considered obvious and indispensable, was not even known before the advent of personal computers.

Even a loaded PC does not come close. Yet, even this progress still falls far short of what is necessary to make a personal computer a congenial tool for thought. Progress has been in the application packages separately, not in ways to integrate.

Consider, as an example of the need for integration, an engineer in his daily activities. He makes calculations, searches tables, standards, textbooks, draft reports, receives and sends mail, retrieves and studies drawings and textual library material, accesses databases (local and remote) and so on. In all these activities separately, computer programs exist that can help. The rapidly falling cost of hardware makes these programs potentially accessible to every engineer. But this is a mixed blessing: if he is to utilize the full potential of all available computer tools, he needs several specialists in attendance, to be available at a few seconds' notice.

Even then he will not be able to use the computer as a truly congenial tool: that is only possible when *no* intermediary is needed. At present he needs an intermediary for many applications because of the complex and idiosyncratic interface provided by the required software. And it does not help that every application package comes with its own, unique programming language, so that two logically identical jobs in different applications need to be done in an exasperatingly different way.

Of course the situation sketched here is not unique to engineers, but is shared by professionals in public or business administration and in scientific research.

What's lacking? A plan for improvement has to be based on a diagnosis of what is wrong. One common diagnosis can be summarized as:

> Computers are difficult to use because *they are not enough like humans.* To make progress we must make them more like humans. Therefore, before we work directly towards a congenial tool

> for mental work, we need progress in Artificial Intelligence.

But another diagnosis is possible:

> Computers are difficult to use because *they are not enough like automobiles*, that is, they are not a tool that one can easily learn to use as an extension of oneself. To make progress towards a congenial tool for mental work, we need to work on the ergonomics of interfaces to software.

The Japanese Fifth Generation Computer System project [Moto-Oka 1982] is based on the first diagnosis. I will argue that to make most rapid progress towards CATMAN we must work on ergonomics rather than on Artificial Intelligence. Moreover, that via ergonomics progress is predictable and will be rapid, as it will be a matter of elaborating existing software technology. In comparison, progress in Artificial Intelligence seems unpredictable: the required results may indeed be around the corner, or it may be a long time before they materialize (if at all).

5 Improvement is a matter of ergonomics, not AI

We saw that what stands in the way of CATMAN can be diagnosed as either a problem in Artificial Intelligence or as a problem in mental ergonomics. There are two episodes from the past that should help in deciding which diagnosis is more fruitful.

In the late 1940's influential administrators perceived an acute shortage of experts available to translate Russian scientific publications into English. In that period there existed considerable optimism about the feasibility of fully automatic high-quality translation, resulting in several well-funded research projects. Lack of progress in the fifties, combined with devastating criticism [Bar-Hillel 1964] of the scientific basis of machine translation, caused funding to be withdrawn.

Let us consider two possible reactions to this failure to get computers to alleviate the shortage of translators.

Reaction 1: The funds should have been spent on Artificial Intelligence. The consensus that emerged in the fifties and caused the demise of projects aiming at fully automatic high-quality translation, was that the text to be translated had to be understood, at least to a certain extent, by the translating agent, human or machine. Machine translation was therefore seen as a problem in Artificial Intelligence. Getting a computer to help in translation was therefore premature — progress in Artificial Intelligence was needed first.

Reaction 2: The funds should have been spent on ergonomics. In the fifties, when the objective was to use computers to help alleviate the shortage in translators, the technology available to translators consisted of a typewriter (electro-mechanical at best) and some well-thumbed reference books. In the early eighties, after machine translation had long been forgotten, and in response to different pressures, there evolved a set of computer tools that have enormously increased the productivity of translators: word processors, spelling checkers, thesauruses, dictionaries, checkers of style and diction. Such software could have been built soon after 1960 when the first time-sharing systems became available.

Thus in 1960, when it was clear that the approach to machine translation taken in the fifties was doomed to failure, it would have been possible to go on to achieve great increases in productivity at low cost. Instead, *it was concluded that the least tractable stage of translation was to remain to receive top priority and that research in Artificial Intelligence was to be motivated in part by the desirability to use computers to increase productivity of translators.*

A lesson to be learned from this episode. There is a similarity between the situation now, in which we suspect that computers can do more to help mental work than is actually the case and the situation in the fifties when it was hoped that computers could help in translation. In the case of translation, the least tractable aspect of the work was selected, leading to Artificial Intelligence. In retrospect, more tractable, even mundane, aspects (namely ergonomics) could have been selected with great success, not only to increase the productivity of translators, but of other office workers as well.

Similarly, when considering how to make a computer into a congenial tool for mental work, there seems to be a great temptation to fall into the same trap: to view Artificial Intelligence as panacea.

To end this section on a positive note, I will conclude with an episode from the past where the right alternative was selected. There was a time when automobiles were difficult to use, for several reasons: for example, because of frequent need for tuning, maintenance, and repair. At that time, a Plutocrat requiring transportation solved this problem by retaining a chauffeur and a mechanic (ideally, but not necessarily, the same person).

When considering obstacles preventing more widely available transportation by automobile, the following diagnoses are possible:

1. build robot chauffeur-mechanics
2. make automobiles easier to use, so that the chauffeuring can be done by the person to be transported and so that only an occasional visit to a garage is required for tuning, maintenance, and repair.

The Japanese FGCS project[3] has selected an alternative in the spirit of the first [Moto-Oka 1982].

6 Logic programming meets the requirements of Mental Ergonomics

In this section I review some of the basic principles of Mental Ergonomics and comment how logic programming can help implement them in CATMAN: *Avoid trying to do two things at a time, Allow the user to do the same thing in the same way (if desired), Exploit useful conservatism*, and *Avoid harmful conservatism.*

Avoid trying to do two things at a time. It is bad ergonomics to define a programming language in such a way that the declarative and the imperative aspects of programming are not easy to separate. Rather than to attempt to define these aspects, I will illustrate them by the example of computing an arithmetic expression using register-to-register machine operations.

Two tasks have to be distinguished here:

- To make sure that the correct expression is evaluated (*what* is computed; this is the *declarative* aspect of programming). What the correct expression is, is only determined by the application, independently of the machine on which the computation is to be performed. Thus, this task can also be thought of as that of solving the *application problem.*
- To determine the sequence of register operations and transfers required to get the correct value in the desired location (*how* it is computed; this is the *imperative* aspect of programming). This task contains the above task. What belongs to this task over and above the application problem is to control the machine. To get this additional aspect right is to solve the *control problem.*

The example of arithmetical expressions is useful because every programming language allows these to be evaluated without having to solve a control problem. Thus, every programmer, at the level of assembler language and up, is familiar with declarative programming. The problem with conventional languages is that this type of programming is only possible with arithmetic expressions, on which the programmer spends only a small proportion of his time. Most of the work requires areas of the language where the application and control aspects of the task are intimately intertwined. As a result, it is possible for an error in control to cause a wrong answer. As a result, a programmer in such a language is forced to try to do two things at the same time.

Logic as a programming language allows a decomposition of an algorithm into what Kowalski [1979] calls its *logic* component (corresponding to the declarative aspect) and its *control* component (corresponding to the imperative aspect). A consequence of Kowalski's approach is that the declarative and imperative aspects are separated, so that *an error in control cannot cause an erroneous answer to appear*; at worst it will cause failure to find an answer. The advantage is that there is no need to solve the application and control problems at the same time.

Allow the user to do the same operation in the same way (if desired). In the existing personal computer systems, the closest approximations to CATMAN require the use of separate programming languages for databases, spreadsheets, intensive numerical computation, system programming, document preparation, the shell, and perhaps even other ones. This means that the same operation (such as procedure and data declaration, procedure call, case selection, iteration, and so on) has to be done in a different way in each of these different languages, violating a principle of ergonomics.

It has been shown that logic programming can be the basis of many different types of programming language: functional [Cheng *et al.* 1990], imperative [van Emden 1976, Rosenblueth 1989], object-oriented [Davison 1988], stream-oriented [Taylor 1989], as well as for database querying [Ceri *et al.* 1990]. Of course, within this framework there are still many opportunities for violating the ergonomic principle by undue proliferation of variety. But by using logic as common framework for whatever different languages are needed, improvement is made easier.

Exploit useful conservatism. A *conceptual interface* represents a beneficial kind of conservatism: it is an interface modelled on a familiar concept so that the known operations on the concept can serve as model for the computer operations that need to be learned. The prototypical example of a conceptual interface is the WYSIWYG editor, where the familiar concept is a sheet of paper. Examples of conceptual interfaces that fit well in logic programming are: *the conversational partner, the pocket calculator, spreadsheets and tables*, and *the filling in of blanks.* I elaborate on these below.

Lack of a conversational partner as conceptual interface can lead to the kind of frustration eloquently voiced by John McCarthy, who complained[4] that even to get a computer to acquire a simple symbol manipulation skill

[3] In its original 1982 version [Moto-Oka 1982]. What the project has actually done since then is more sensible. In fact, they have been a prolific contributor to logic programming, my proposed technical basis for CATMAN. But the preoccupation with parallelism and with big machines remains, and this can only be traced back to the initially intended role of Artificial Intelligence.

[4] In debate with Sir James Lighthill on BBC TV in 1974.

is like having to perform *brain surgery*. He explained that the goal of his research is to program computers in such a way that one just needs to *tell* them. That is, to model the interface on that of a conversational partner.

This ideal has been realized to a certain extent by MYCIN [Shortliffe 1976], an early expert system. The user interacts with it in the following way. If the user lacks information, he *asks* a question to which the machine may respond with an answer. If the user knows something that the computer doesn't, he *tells* a fact, or a rule. If the user is puzzled by an answer, then he can *request an explanation*, which comes in the form of facts and rules chaining the facts to the answer. Sergot [1982] and Shapiro [1983], have shown that this conceptual interface finds a natural home in logic programming. They added to the initial version embodied in MYCIN the possibility of making the computer and user play symmetrical roles.

The programming language LISP is an example of a conceptual interface, albeit in an inverted way. The basic interaction mode in LISP does not need to be learned because it is the same as that of a pocket calculator: enter expression to be evaluated, get in return its value. The curious inversion lies in the fact that the familiar concept, the pocket calculator, is of more recent origin than the beneficiary of the conceptual model, namely LISP itself.

In the early days, computers were used in a rigidly planned way. With the advent of time-sharing, users were given the illusion of having a machine of their own, allowing in principle an intimate, interactive, and spontaneous use. Software to exploit this possibility was slow in coming: only with the advent of programs modelled on the spreadsheet as conceptual interface for personal computers has this mode of use been convincingly demonstrated. A similar, but significantly different, interface is that of a *table*, where rows and columns play different roles. Both interfaces have been shown to be compatible with logic programming [van Emden *et al.* 1986, Cheng *et al.* 1988].

Filling in the blanks of a form is a useful, though not widely loved, conceptual interface. It has been exploited in Query-By-Example [Zloof 1977] to provide one of the more congenial query languages for databases. The queries of the logic programming language Prolog are similar [van Emden 1977, Kowalski 1979].

Avoid harmful conservatism. Exploiting a conceptual interface is a useful form of conservatism. Insisting that only English is fit for humans to communicate with our tools is not. The optimism about the utility of natural language for a user-computer interface is based in no small degree on the work of T. Winograd [1972], who himself, however, subsequently made the following observation [Winograd and Flores 1987]:

> The practicality of limited natural language systems is still an open question. Since the nature of the queries is limited by the formal structure of the data base, it may well be more efficient for a person to learn a specialized formal language designed for that purpose, rather than learning through experience just which English sentences will and will not be handled. When interacting in natural language it is easy to fall into assuming that the range of sentences that can be appropriately processed will approximate what would be understood by a human being with a similar collection of data. Since that is not true, the user ends up adapting to a collection of idioms – fixed patterns that experience has shown will work. Once the advantage of flexibility has been removed, it is not clear that the additional costs of natural language (verbosity, redundancy, ambiguity, etc.) are worth paying in place of a more streamlined formal system.

An interface where the user is confronted with seemingly random breakdowns and has to guess at what will work and what won't, is frustrating and inefficient — bad ergonomics.

A special-purpose notation can not only be a convenience, but even a genuine augmentation of the intellect. Such a notation should be seen as evolution of language, helping further development of the intellect. *Such co-evolution of language and intellect should be allowed to continue in the computer age and should not be stifled by doctrinaire insistence that only English is fit for humans.*

"Natural, easy-to-use" interfaces are to be approached warily when they are slower in use than other interfaces. Windows and a mouse can be extremely enticing when a novice finds that already after the first half hour he can get simple jobs done on a computer. But an interface that takes ten times as long to learn and allows the user to work twice as fast is worth the extra trouble after four and a half hours of use. As most users spend over a hundred, or even over a thousand hours with a computer every year, it is clear that preference for the "natural and easy-to-use" can be a form of harmful conservatism.

7 Concluding remarks

If a moratorium on hardware improvement were to go into effect today, it would take decades before software caught up far enough to exploit hardware to a reasonable extent. Such a degree of exploitation includes the use of a computer as a congenial tool for thought, and deserves to be primary focus of computer science.

To exploit the potential of computers as tools to augment the intellect, the Fifth-Generation Computer Systems Project has relied on expected advances in Artificial

Intelligence. Experience in attempts to use computers to increase productivity in translation between texts in natural language suggests that more mundane approaches, summarized under Mental Ergonomics, are more effective.

I have argued against the use of Artificial Intelligence and of natural language processing by computer. Lest I be misunderstood, let me emphasize that this concerned the particular applications addressed in this paper. Artificial Intelligence as cognitive science is as interesting and important as particle physics and cosmology. Natural language processing by computer has by now reached the stage where it can be a valuable aid to human translators, and to authors more generally. This is consistent with the reservations quoted from Winograd.

Acknowledgments

Generous support was provided by the British Columbia Advanced Systems Institute, the Canada Natural Science and Engineering Research Council, the Canadian Institute for Advanced Research, the Institute of Robotics and Intelligent Systems, and the Laboratory for Automation, Communication and Information Systems Research.

References

[Bar-Hillel 1964] Y. Bar-Hillel. *Language and Information.* Addison-Wesley, 1964.

[Bush 1945] Vannevar Bush. As we may think. In Adele Goldberg, editor, *A History of Personal Workstations*, pages 237–247. Addison Wesley, 1988. First published in *Atlantic Monthly*, July 1945.

[Ceri *et al.* 1990] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer, 1990.

[Cheng *et al.* 1988] M.H.M. Cheng, M.H. van Emden, and J.H.M. Lee. Tables as a user interface for logic programs. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, pages 784–791, Tokyo, Japan, November–December 1988. Ohmsha, Ltd.

[Cheng *et al.* 1990] M.H.M. Cheng, M.H. van Emden, and B.E. Richards. On Warren's method for functional programming in logic. In David H.D. Warren and Peter Szeredi, editors, *Logic Programming: Proceedings of the Seventh International Conference*, pages 546–560, Jeruzalem, 1990. MIT Press.

[Davison 1988] A. Davison. Polka: a parlog object-oriented language. Technical report, Department of Computing, Imperial College of Science and Technology, University of London, 1988.

[Engelbart 1963] D.C. Engelbart. A conceptual framework for the augmentation of man's intellect. In P.W. Howerton and D.C. Weeks, editors, *Vistas in Information Handling*, pages 1–29. Spartan Books, 1963.

[Engelbart 1988] Douglas Engelbart. The augmented knowledge workshop. In Adele Goldberg, editor, *A History of Personal Workstations*, pages 187–232. Addison Wesley, 1988.

[Horton 1982] S. Horton. *Thinking Through Writing.* Johns Hopkins University Press, 1982.

[Kowalski 1979a] R.A. Kowalski. Algorithm = Logic + Control. *Communications of the ACM*, 22:424–436, 1979.

[Kowalski 1979] R.A. Kowalski. *Logic for Problem-Solving.* Elsevier North-Holland, 1979.

[Licklider 1960] J.C.R. Licklider. Man-computer symbiosis. In Adele Goldberg, editor, *A History of Personal Workstations*, pages 131–140. Addison Wesley, 1988. First published in *IRE Transactions on Human Factors in Electronics*, March 1960, pp. 4–11.

[Moto-Oka 1982] T. Moto-Oka, editor. *Fifth-Generation Computer Systems.* North-Holland, 1982.

[Rosenblueth 1989] D. Rosenblueth. *Exploiting Determinism in Logic Programming.* PhD thesis, University of Victoria, 1989.

[Sergot 1982] M. Sergot. A Query-The-User facility for logic programming. In P. Degano and E. Sandewall, editors, *Proceedings European Conference on Integrated Interactive Computing Systems.* North Holland, 1982.

[Shapiro 1983] Ehud Shapiro. *Algorithmic Program Debugging.* MIT Press, 1983.

[Shortliffe 1976] E.H. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* Elsevier, 1976.

[Taylor 1989] Stephen Taylor. *Parallel Logic Programming Techniques.* Prentice Hall, 1989.

[van Emden 1976] M.H. van Emden. A proposal for an imperative complement to prolog. Technical Report CS-76-39, University of Waterloo, 1976.

[van Emden 1977] M.H. van Emden. Computation and deductive information retrieval. In E. Neuhold, editor, *Formal Description of Programming Concepts*, pages 421–440. North Holland, 1977.

[van Emden *et al.* 1986] M.H. van Emden, M. Ohki, and A. Takeuchi. Spreadsheets with incremental queries as a user interface for logic programs. *New Generation Computing*, 4:287–304, 1986.

[Winograd 1972] T. Winograd. *Understanding Natural Language.* Edinburgh University Press, 1972.

[Winograd and Flores 1987] T. Winograd and F. Flores. *Understanding Computers and Cognition.* Addison-Wesley, 1987.

[Zloof 1977] M. Zloof. Query-By-Example: a database language. *IBM Systems Journal*, 16:324–343, 1977.