

SOME IDEAS ON THE EDUCATIONAL USE OF COMPUTERS

Harald WERTZ
L.I.T.P. / CNRS
2 Place Jussieu
75005 Paris

&

Université Paris VIII
Dépt. Informatique
2 rue de la Liberté
93526 St-Denis Cedex 02
FRANCE

Abstract

In this paper we will develop a new point of view on the educational use of computers, stressing its possibility for interactive use and the intellectual implications of the programming activity. After giving a rather detailed analysis of programming viewed as a cognitive process we propose a dual model centered on the notions of *piloting* and *programming*, followed by some snapshots of the interaction of a child with a computer. We will end with a proposition for the construction of an intelligent and convivial user environment.

INTRODUCTION

Computers in education have traditionally been used as a tool for scholarly research. An as yet unrecognized but logical use in today's academic environment is the proposed use of computers as an interactive environment for exploring the intellectual processes.

Basic Ideas

Several fundamental observations form the basis of this new approach.

- 1) historically, the computer has been the first machine developed which automates intellectual processes.
- 2) presently only those aspects of intellectual processes which are humanly understood can be executed by a computer.

- 3) reading a computer program which models a particular process or knowledge permits one to infer the modelled process or knowledge.
- 4) the construction of a computer program forces an understanding of the application domain of the program.
- 5) a computer program is the formalization of a problem *and* its solution. This formalization is operational, i.e. it is testable, executable and verifiable. In addition, it is dynamic, i.e. subject to continuous modification, parallel to the development of the implied knowledge.
- 6) computer programming, in an interactive environment, permits one to become conscious of the mechanisms of the problem solving behaviour involved.

COMPUTERS AND INTELLECTUAL PROCESSES

In this paper, when we refer to computers, we refer not to the set of electronic and mechanical devices which form a computer installation, but exclusively to their applications and their content - the software.

If we examine the different machines and tools that have been adapted for educational purposes (the typewriter by Freinet, the mechanics of automobiles by Dewey, the video-set by Illitch, etc.), we realize that each of them contains applications of knowledge in restricted *particular* areas. For example, it is reasonable to demonstrate in examining automobile motors the concepts of mechanics, but it is completely impossible to convey, through manipulation of automobile motors, the concepts of quantum physics, problem solving, psychology, sociology etc.

On the other hand, the computer is - as demonstrated by Turing - a general tool, that is to say, without theoretical limitations (except those we ourselves introduce).

In order to demonstrate this point we will first define what we mean by *program* and *programming*.

Program and Programming

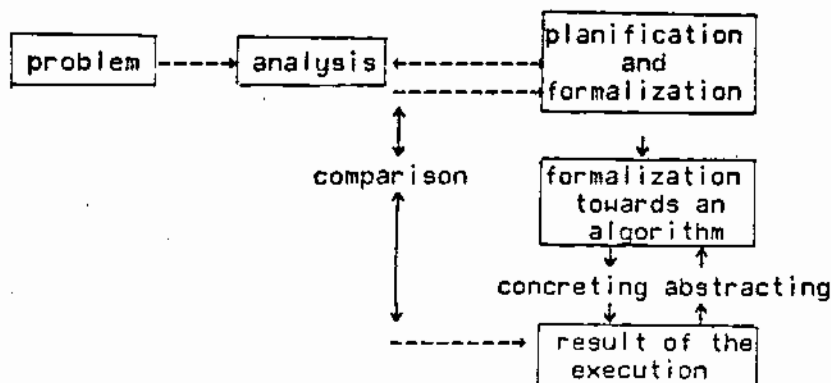
From the point of view of a computer, a program is a sequence of intelligible and executable statements.

From the point of view of a programmer, a program is the expression of the understanding of a given problem, and, since this understanding may develop and change, the program is rarely a finished product and therefore, is subject to modifications, improvements and revisions.

Let us explore this point: the construction of a program may be decomposed in the following way:

- 1) *choice of a problem.* In order not to bias the use of the computer (to respond to the wishes and interests of the student), we should ensure that the student is able to freely choose the problem to be solved by the machine.
- 2) *analysis of the chosen problem.* This activity implies understanding the problem, decomposing it into smaller subproblems, defining the relations between the different parts of the problem, defining a language for the analysis of the problem, abstracting unimportant details and recognizing the principal parts, classifying, perhaps working by analogy, using diagrams, etc ...
- 3) *planning a solution.* To reconstruct the different parts into a whole, to define an order in which the different parts have to be solved, sometimes even to develop a strategy.
- 4) *formalizing the solution.* The transcription of the plan and the results of the analysis in a formal language (apparently a programming language). This includes formalizing the representation of the given data.
- 5) *execution of the program.* The result of the formalization is the construction of a program. It has to be tested, i.e. executed with concrete data.
- 6) *analysis of the results.* Even experienced computer scientists seldom have programs without errors (irregularities or bugs). One has to compare the given results with the desired ones (which implies that the pupil has chosen his examples well, neither too complex nor too simple). The permanent possibility of having errors in the program is considered extremely important for our educational model. We will come back later to this point.
- 8) *modification of the program.* Once one has found bugs or irregularities in the program, it has to be modified. But - and this is one of our main points - as the initial program has been constructed as a result of the problem analysis, the irregularities of the program can immediately be rethought as insufficiencies or errors of the analysis process, henceforth of the understanding of the problem.

Schematically we have:



- 2) the learning of an instrument is happening at an individual rhythm, different for each apprentice, and - last but not least -
- 3) the apprentice has the opportunity to determine his own projects, i.e. which piece of music he wants to learn.

Here we will stop our analogy and return to the programming activity.

Piloting and Programming

Given that we want the pupils to program, and given that programming is a very complex activity, we propose a mode of learning which reveals the local simplicity as well as the global complexity of programming. We will call this mode *piloting*. Three preliminary characteristics of the programming environment are obligatory:

- 1) the programming language must permit interactive use of the computer.
- 2) the computer on which the pupil is working must include at least one peripheral device whose actions are understandable and evaluable instantaneously. It does not matter if it is a graphic display, a music-box or even a mobile robot.
- 3) the programming language must contain a set of simple orders that will permit a change of state in the peripheral device(s).

The pupil, possessing the indispensable rudiments of the language (i.e. after having read or being explained the few instructions which modify the state of the peripherals), will be invited to explore interactively the possibilities offered by the machine.

To be more explicit, let us take as an example the LOGO programming language [12]. This language has a set of instructions which direct a mobile vehicle, called *turtle*, which can trace its own movements by means of an incorporated pencil. The instructions we will consider are:

- | | |
|---------------------------------|--|
| FORWARD n: | moves the turtle n units forward |
| BACKWARD n: | moves the turtle n units backwards |
| RIGHT n: | changes the direction of the turtle n degrees to the right |
| LEFT n: | changes the direction of the turtle n degrees to the left |

We immediately recognize that each of these instructions is extremely simple and immediately understandable.

During *piloting* mode, the pupil explores interactively the possibilities. For example, he will discover that the instruction FORWARD 10, really makes the turtle advance 10 steps, all the

Therefore we consider the construction of a program as the exact expression of the understanding of a problem; to reiterate - it is the formalization of the problem *and* of its solution. This operating (executable, testable, verifiable) formalization represents the educational power of the programming activity: the execution of the program and its results reflect the understanding of the given problem.

The abstraction of the algorithm is translated to a tangible and understandable result. The difference between computed results and desired results (the irregularities or bugs) can be understood by the pupil himself, and can be transformed as an improved algorithm.

The programmer becomes the teacher of the machine. As such, the position of the pupil in the academic environment is inverted: from someone taught he becomes a teacher and verifier of his own product. The power of command over the machine is (observably) extremely stimulating, and obliges a deep understanding of the problem under attack. The pupil, in programming his projects is more than a passive user of a program. To the contrary: he is its active creator and the engineer of its maintenance.

Local Simplicity vs Global Complexity

Every person who programs realizes an astonishing discordance between the local simplicity of a program on the one hand and the - often unsurmountable - global complexity on the other hand. Locally each instruction is immediately understandable, but the global structure of a program appears rather complex: too much interaction between different parts, too much bookkeeping, etc.; often the surprised user finds it difficult to understand what is really happening.

Though this remark is evidently in relation to the programming language used, it seems that this complexity originates mainly in the complexity of the problems considered. (The differences of the language only result in the fact that this complexity is reached more or less quickly.)

Let us insist: programming is a very difficult activity. But so are other disciplines: mathematics, linguistics, music, etc. The problem is not how to make learning to program easier but how to make it efficient and interesting.

The first answer to this question that comes to mind is to say : well, it should be pleasant and stimulating.

There can exist other pleasant and stimulating learning environments: for example, learning a musical instrument. No-one will pretend that this is an easy task, but, nevertheless, children may learn very fast to play an instrument. We think some of the most important factors for this rapid learning are:

- 1) the immediate feedback, the apprentice can evaluate for himself whether or not he made an error,

while drawing a line, and that the instructions RIGHT 90, FORWARD 10, RIGHT 90, FORWARD 10, RIGHT 90, FORWARD 10, executed consecutively will make the turtle draw a square.

The important points in this mode are first, the pupil's ability to discover by himself the necessity of more complex instructions (for example instructions for selection or repetition) and second, the discovery that each of the instructions is in itself very simple, but as in the *incremental* drawing obtained by the successive movements of the turtle, their composition may become very complex extremely quickly.

The moment the pupil has understood a sequence of instructions and stops repeating it, he can switch to *programming* mode and group the instructions together to construct a new command. (Let us remark that the absence of this possibility to construct new commands is one of the inadmissible limitations of the BASIC programming language).

Once in programming mode, the piloting mode is still available - to test some instruction, or to build some hypothesis for a particular piece of program, or to try to ...

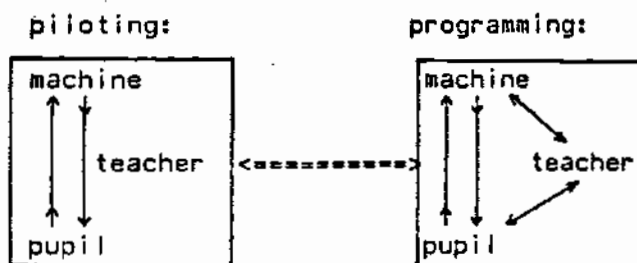
This mode of programming permits the pupils to feel immediately comfortable with the computer. It permits them to visualize particular aspects of the program to concentrate on errors, or to verify hypotheses or algorithms.

The two modes are complementary: the program is constructed once the pupil has well understood the problem by using piloting to better grasp the problem.

- Summary of the Proposed Programming Activity

In the preceding paragraphs we have seen the fundamental difference between our described approach (autonomous learning) and the more classical approaches.

If we consider the case of one pupil using one computer we have the following two schemas:



with a permanent switch between those two modes of activity (which is expressed by the double arc between the two frames).

Note that, as we have proposed, the computer will be used both for *ascending* (= piloting) as well as *descending* (= programming) modes of working.

Let us mention that the piloting mode can also be an extremely useful process for understanding programs written by others. In spite of passively reading the program, this mode permits one to explore it interactively (i.e., to test it, etc.) - to read it in a basic active way. Therefore one is able to not only understand what the program is doing, but how it does it as well.

Piloting, Programming and Mechanisms of the Mind

Until now we have referred only to points 1 through 5 outlined at the beginning of the paper. We have left the most fundamental and stimulating aspect of the programming activity for the final part of the paper.

6) computer programming, in an interactive environment, permits one to become conscious of the mechanisms of the problem solving behaviour involved.

The inferences of this statement are: First, that there exists a deep parallelism between programming and problem solving, i.e. intellectual operations.

Recalling our characterization of programming: the construction of programs implies a continuous and deep analysis of the problem to be solved, its decomposition into sub-problems, thinking by analogy, generalization of some necessary concepts, abstraction to the level of formalization, solution of partial problems, their synthesis into a global solution, analysis, and correction of errors.

These points are basically those enumerated by psychologists and scholars when writing about human problem solving behaviour [6-8, 14]. The difference is that computer programming makes them become explicit and models them.

We will illustrate this by playing, for a moment, a game of M. Minsky [10] while enumerating some common concepts of computer science :

symbol table	list structure	backtracking
interpreter	interruption	procedure
iteration	data-type	heuristique
recursion	micro-program	matching
stack	time sharing	garbage collection
...

All these concepts are extremely familiar to programmers *and* to cognitive psychologists. Following Minsky, these concepts of computer science are so new that they qualitatively changed the understanding of the human mind. The ability to describe complex mental processes was strengthened by taking into consideration these standard programming concepts. This consideration of computer science as part of theoretical psychology is also supported by A. Newell and H. Simon [11].

The concepts of computer science are more than simple metaphors of our intellectual functioning: they give a possible explicit description of it.

Coming back to our claim that programming permits one to consciously understand the mechanisms of the mind, note that the adequate interactive environment is (to our understanding) basically this environment of piloting and programming. The understanding comes from self-direction and by efforts to verbalize one's experiences (either with co-pupils, or with the teacher), to communicate them and to explain them.

The activity of programming, as an independent intellectual agent, will permit the generalization and the transfer of the methods manipulated and discovered.

Note that up until now, all experiments with children in this direction have shown considerable results. Progress has not been directly measured by school tests, but has been exhibited in the children by an augmented awareness of intellectual processes and by the fact that - not as concluded S. Papert, that they have learned to learn - but that they have learned not to be anxious of learning.

CONCRETE ASPECTS

Here we will discuss some extracts from a protocol [9] of one of the first sessions with a computer by a child from a SES. SES's are, in France, special schools for children having problems in keeping up with the normal classes. This will serve to briefly illustrate the mechanisms involved in autonomous learning.

Some Preliminary Remarks

We have chosen this example for the following reasons:

- 1) it is taken from an experiment which continued for a sufficiently long time (2 years),
- 2) the experiment was conducted within the normal school setting in collaboration with the teacher,
- 3) the experiment is conducted with children having severe learning problems.

The exact environment is described in [9]. The particularly significant aspects are discussed here.

Description of the Activity of the Children

During the very first sessions, the children get acquainted with the programming language and the machine used. In our case the language is VLIISP [2, 4] augmented by the graphic instructions of LOGO [15]. This introduction is done interactively, between

child and computer. Here the child discovers the importance of some special keys, such as the RETURN and DELETE key, and the effects of following the basic instructions. The child is invited to develop an experimental method, eventually trying different versions in order to choose the one he thinks is best.

A typical example of this experimental method (generating mini-theories corresponding to the problem being worked on) is the method developed by a group of children in order to understand the difference between *name* and *value*, two fundamental concepts of symbolic reasoning.

Here is the problem: in every programming language there exist special symbols for determining if a given expression needs to be evaluated (if you have to compute its value) or not (if you can take it as a literal). In LISP this symbol is the prefix "" (quote character). The problem for these children was to be able to distinguish when they needed this prefix and when it wasn't necessary. They systematically wrote two versions of the program. One using and one not using the quote-character. Their analysis of their results permitted them to then understand the difference between the two versions as well as the meaning and necessity of these types of constructs. The newly acquired knowledge was then used to build more complex expressions.

This example begins to show the positive and personal nature of the acquired knowledge. What the children learn is no longer some abstract knowledge, such as multiplication and division tables for example, but knowledge deduced and discovered by specific needs: the realization of a personal project.

This small example serves to illustrate also the fundamental role of *piloting* as a process for experimenting, discovery, and the construction of a working hypothesis to be immediately validated or invalidated by the man-machine interaction.

Extracts of a Protocol

For these children the learning and the practice of programming isn't just a task to learn, it is a way to teach them to

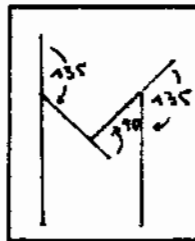
- analyze problems
- form abstractions
- formalize solutions
- divide the solution of a problem into a set of solutions for its subproblems
- develop the habitude of verifying general solutions with particular cases
- consider errors not as disastrous, but as temporary obstacles which need to be overcome
- develop a constructive self-critical behaviour.

Here we will examine "A"'s session with the computer. "A", a 12 year old, had chosen the task of making the machine write her name in large letters on the colour display connected to the computer. She had chosen this problem by herself, as a result of her first understanding of the graphics possibility of the programming language.

During the analysis of the problem - carried out independently of all influence by the teacher - she had already divided this problem into smaller subparts: make the computer draw the letters composing her name. Afterwards she had generalized the problem: first teach the machine how to draw each letter of the alphabet, then use the different solutions to write any name you want.

Two extremely powerful methods of problem solving have been used here quite naturally (and on a problem which involved "A" personally) : first, the recognition that the problem actually constituted several subproblems followed by the generalization of the subproblems - modifying her original problem. She had, as notes Papert in a similar case [13], rediscovered one of the greatest sources of intellectual power: the hierarchical construction of new concepts in using those already known.

For the session we are looking at, she had prepared her first letter: M. She had carefully drawn it, calculated all the angles

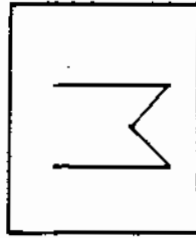


and prepared the following program

```
FORWARD 10  
RIGHT 135  
FORWARD 5  
LEFT 90  
FORWARD 5  
RIGHT 135  
FORWARD 10
```

Remember that the children from SES's have a very low academic level (corresponding to elementary school) and that one of their problems is just the (pretended?) inability to work autonomously and to clearly analyze the given problems.

The execution of this program fragment drew a very good M, but on its side:



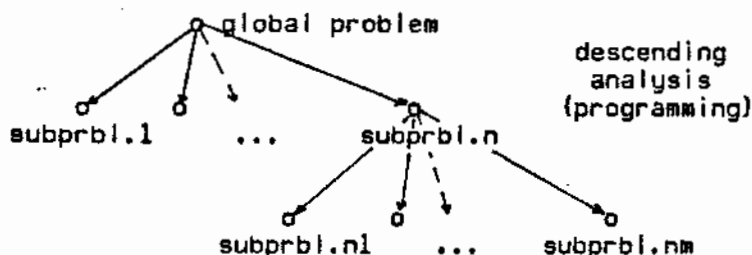
Here is another important point: the computer almost always gives a result that generates a feeling of success: "A" could *see* the result of her work and she could understand and evaluate it. In programming you never have "wrong" results, there are only results which aren't correct *yet*. The difference is fundamental: there is no disbelief in one's capabilities (generated by classical remarks as "that's wrong"), but a concentrated search to find the reasons for the irregularities in order to eliminate them.

In this case, the reason for the bug is evident; there isn't an error in the set of instructions drawing the M (it really has the shape of an M) but in the preconditions: the initial direction of the turtle is not good. So "A" introduced a preliminary instruction, LEFT 90, to make the turtle first turn in the right direction. This gave her a fine vertical M.

Note the complete autonomy of "A". Until now she has worked without any interaction from the teacher (neither to correct nor to criticize). We point out that autonomous doesn't mean isolated. "A" continuously communicated her activities, thoughts and success to the teacher and her co-pupils. The role of verbalization is considered very important.

Once terminating piloting mode with the correct series of instructions, "A" decided to save these instructions for later use in writing her name. She wanted to translate them into a function for use in programming mode.

Reexamined, the difference between the two modes of working is, in a descending manner, decomposition into a series of subproblems. Each of the solutions to the subproblems is a program or procedure. The solution of the global problem is a super-procedure which calls the different sub-procedures.



Often the solution to subproblems are constructed in an ascending style in piloting mode for constructing the solution interactively and experimentally (as was the case here with the letter M).

In order to ensure that the use of the computer in the schools won't be as a new gadget in an ancient culture [13], it is paramount to know that it represents the first general purpose tool to permit pupils to constructively use their knowledge to build, to test, and to modify their own theories. The computer can form a learning environment richer in possibilities for self-discovery than any other known educational tool.

CRITICS AND PROPOSITIONS

There remain a lot of questions concerning the method and the effects of such learning. What will be the intellectual and emotional behaviour of a child immersed in such an environment for a rather long period of time, and how can we facilitate the transfer of the acquired knowledge, are two fundamental questions. However, we feel the results obtained so far validate our optimism in this approach.

What we are primarily interested in is the learning of basic computing principles as a way of forming a model of the working of our minds and as a process for acquiring knowledge. This is not just learning a programming language. The second point we are interested in is the construction of learning environments. We are optimistic that the most powerful tool available for these environments is the computer.

The Programming Languages

The teaching of problem solving concepts in using the computer is completely different from the teaching of programming languages. In one case we construct step by step the universe of programming and of computers. In the other, the most powerful instructions may have already been introduced at the beginning (graphic instructions are powerful instructions). Programming is not a goal but a vehicle for the transmission of concepts (i).

The awareness of fundamental concepts, learned by programming, seems to be independent of the syntax of a particular language. (i) The dependence is on the semantics of the instructions of the language, i.e. of the expressive power of the instructions. The more the language is semantically powerful, the more the children progress.

(i) Don't misunderstand: we aren't at all excluding computer science courses in the schools. To the contrary, computer science is a scientific discipline like mathematics, physics, etc., which *must* be taught in schools. What we are discussing here is the educational value of computers; their use as a tool for learning and the potential they offer to change, in a very fundamental way, the academic methods of learning.

There remains the question of development of programming languages: those we are using now are *action languages* at a very high level. But they lack *descriptive* elements. It is extremely easy to express processes, activities, in these languages, but as soon as one has to represent complex data, such as networks or scenarios, the resulting expressions become rather unreadable. Artificial Intelligence researches have developed many very high level languages (KRL, FRL, Smalltalk, Plasma, to list just a few), as well action oriented as description oriented. In order to give the pupils in schools the most powerful languages, we should integrate all the useful aspects in a unique language.

Note that the problem of language applies not only to the educational applications of the computer, but to its use by naive users in general.

The Computer Environment

Every programming activity needs an environment, not just hardware - this will be the subject of another paper - but also software: the programmer needs the ability to read files, to save his programs, to edit them; he needs the ability to interrupt the execution of his program in order to check the state of variables etc.,

The success or the failure of a session with a computer depends on the availability of reliable hardware and software - this is obvious for use by an experienced programmer, but is particularly necessary for the naive programmer.

The computer systems actually used in schools (LSE in France, BASIC in most places in the USA) lack an adequate environment. What we find are only the absolute minimum: restricted text editors, working at the level of a single line of a program (no-one perceives a program as a sequence of lines!), and rudiments of a filing system.

What is needed is an *interactive* and *intelligent* environment, which can be the fundamental component of every future computing system. We should develop text editors to work at the conceptual level of the program - not at the character or line level. This needs the preliminary development of automatic program understanding and documentation systems. Programming should be done in collaboration with the machine: the computer should *help* in indicating and correcting simple errors [3, 16], instead of

(i) The language we use is VLISP [2, 4] augmented by some graphics [1, 15]. The results obtained with the children are almost identical in those experiments using VLISP and those using LOGO [12]. As reported in [17], we have seen one child switching languages in the midst of a project without being disturbed by the syntactical differences.

delivering esoteric error messages. (i). The machine should give indications as to what one can do to correct the error and even give examples if necessary. It should be as useful as a *dynamic* manual. We need systems capable of helping intelligently during the running and the modification of the programs.

A good programming environment should permit multiple representations of a program or data. It should be of assistance to the programmer.

Different systems incorporating these characteristics already exist as a result of artificial intelligence research. It is true that these systems are only running on large computers. But the development of microprocessors is going on at such a high rate that the first small computers capable of supporting such a *convivial* environment are already being made available commercially.

One last point: Many people are developing "simple" programming languages specially designed for learning purposes. The most widely known example is BASIC. But these languages are in reality very *hard to manipulate*. Their "simplicity" is only apparent, it is abstract and dissolves in *practice*. The tools that are the most simple to manipulate are almost always the tools that are the most sophisticated.

ACKNOWLEDGEMENT

The English used in this paper has considerably benefited from the advice and the careful reading of Patte Wood and Harlyn Baker.

REFERENCES

- [1] AUDOIRE, L., *COLORIX*, memoire de maitrise, Dept. Informatique, Universite Paris 8, 1976
- [2] CHAILLOUX, J., *Le modele VLISP: Description, Implementation et Evaluation*, These, Universite Paris 6, 1980
- [3] GOOSSENS, D., *Meta-Interpretation of Recursive List-Processing Programs*, IJCAI 7, Tokyo, 1979
- [4] GREUSSAY, P., *Contribution a la Definition, l'Interpretation et a l'Implementation des Lambda-langages*, These, Universite Paris 7, 1977
- [5] GREUSSAY, P., *Aides a la Programmation en LISP - Outils d'observation et de comprehension*, AFCET, Cargese, 1979

(i) to be convinced of the stupidity of actual error messages, just take any commercial micro computer and type in something containing an error. Usually the machine just answers: what?.

- [6] KILPATRICK J. & WIRSZUP I., *Soviet Studies in the Psychology of Learning and Teaching Mathematics*, vol. 1-4, University of Chicago Press, 1969
- [7] LOMPSCHER, J., *Geistige Erziehung als Forderung der Zeit*, Ernst Reinhardt Verlag, Munchen/Basel, 1968
- [8] LOMPSCHER, J., *Verlaufsqualitaeten der geistigen Taetigkeit*, Volk+Wissen, Volkseigener Verlag, Berlin 1976
- [9] MATHIEU, F., *Resume de l'Experience SES*, to be published in: Rapport Simon, La Documentation Francaise, 1980
- [10] MINSKY, M., *Computer Science and the Representation of Knowledge*, in *The Computer Age*, M.I.T.-Press, 1979
- [11] NEWELL, A. & SIMON, H., *Human Problem Solving*, Prentice Hall, 1972
- [12] PAPERT, S., *Computer and Learning*, in *The Computer Age*, M.I.T.-Press, 1979
- [13] PAPERT, S., *A Computer Laboratory for Elementary Schools*, LOGO memo n.1, M.I.T., 1971
- [14] POLYA, G., *How to solve it*, Princeton University Press, 1945
- [15] WERTZ, H., *A System to Improve Incorrect Programs*, 4th Int. Conf. on Software Engineering, Munich, 1979
- [16] WERTZ, H., *Manuel VLISP/LOGO*, rapport technique, Universite Paris 8, Dept. Informatique, 1980
- [17] WERTZ, H., PEROLAT, D., MATHIEU, F., *L'experience d'Arc-et-Senans, rapport final*, RT-11-79, Universite Paris 8, Dept. Informatique, 1979