A psychopedagogical approach to the teaching of a programming language in secondary schools

BRUNO VITALE
Université de Genève et de Fribourg

Abstract

Some of the aspects that characterize the novelty and the difficulties observed in the pupil/computer interaction in the school context are discussed from a psycho-cognitive and a psycho-pedagogical point of view. Special emphasis is put on the need to be particularly sensitive to the problems posed to pupils by the introduction of a formal language and by its use for programming and problem-solving.

Key words: *Formal languages, Programming, Pupil/computer interaction, Secondary Schools.*

Una aproximación psicopedagógica a la enseñanza de un lenguaje de programación en la Escuela Secundaria

Resumen

Se plantean en este artículo, desde un punto de vista psico-cognitivo y psico-pedagógico, algunos de los aspectos que caracterizan la novedosidad y las dificultades que se observan en la interacción ordenador/alumno en contexto escolar. Se hace hincapié en la necesidad de adoptar una actitud receptiva a los problemas que les plantea a los alumnos la introducción de un lenguaje formal y su uso para programar y resolver problemas.

Palabras clave: *Lenguajes formales, Programación, Interacción alumno-ordenador, Escuela Secundaria.*

Dirección del autor: Université de Genève. Faculté de Psychologie et des Sciences de l'Education. 24, rue du Général Dufour. 1205 Genève. Suisse.

Original recibido: Marzo, 1990. *Revisión recibida:* Abril, 1990. *Aceptado:* Junio, 1990

«J'écrirai ici mes pensées sans ordre, et non pas peut-être dans une confusion sans dessein: c'est le véritable ordre, et qui marquera toujours mon objet par la désordre même...»

Pascal, *Pensées*, nr. 373

INTRODUCTION

The teaching of informatics in secondary schools has been motivated, in recent years, by a number of reasons which go from «the need to be prepared to an impending information revolution in society» to «the need to be sensitive to a shift of learning habits from notional (passive) to procedural (active) knowledge». The very number of supposedly *good* (and different) reasons for introducing some teaching of informatics in secondary school curricula is, in itself, suspicious. As a matter of fact, an assessment of the present practice shows that *wishful thinking* generally dominates on *good experimentation, thoughtful analysis and careful evaluation* (Pea & Kurland, 1984; Johanson, 1988; Krendl & Lieberman, 1988; Vitale, 1989).

Most of the research published so far (Vitale, 1988 provides and extended and critical bibliography) has been centered on the technical —e.g., lexical and syntactic— aspects of learning a programming language at school, and on the possible transfer of acquired problem-solving competences to other fields than programming. In the course of a formative evaluation of the teaching of the LOGO programming language to secondary school pupils in Geneva (Hofmann *et al.*, 1987, 1989), we have become aware of the need to investigate more fully a new, essential dimension: the *pupil/computer relation*, in its psycho-cognitive and psycho-pedagogical aspects.

I will briefly discuss, in the present paper, one of these aspects: the *cognitive novelties and obstructions encountered*, by adolescent pupils, *in the acquisition of a formal language*. For a more detailed analysis and for the discussion of a number of psycho-pedagogical suggestions, I shall refer the reader to Vitale (1990).

PROGRAMMING LANGUAGES SEEN AS OBJECTS OF KNOWLEDGE

In a series of interviews with a Swiss editor, Weizenbaum described his human and intellectual career since the forced emigration of his Jewish family, in 1935, from Berlin to the United States. When asked why he chose mathematics when entering college in 1941, he replied that there were several reasons for it, one of them being that «when I went to school first in the United States, I knew no English; so I was able to understand only a little, but one thing I did understand and that was mathematics, which is based on an international language» (Weizebaum, 1984, p. 10).

This requires however that mathematics —or the language of mathematics— be recognized as a *language*, and as a language different enough from the pupil's natural language to become a new cognitive and representative tool. In the same way, an introduction to programming —or to the language of programming, i.e. to at least one of the programming languages— demands that *that* language be recognized as qualitatively different from the pupil's natural language and from the language of mathematics, so as to become a new, legitimate *object of knowledge*.

How can we compare the psychological realities of the mastering of a natural language (NL) as opposed to a formal language (FL)? It is only in a process of interaction and interference between them, in the concrete context of *making use* of both of them, that their functional similitudes and differences will become apparent. The FL itself and the computer response to programming should become an object of knowledge for both pupils and teachers, who are then made attentive to their expectations and to their representations of the computer as an interlocutor. At the same time, they will become aware of the way they make use of a FL to model —on a computer— their experience of reality. This implies that the acquisition of a LF should be *personally constructed* by both pupils and teachers in a slow process of deepening mastery and of possible clashes with the traditional use of their NL (Vitale, 1988, 1990).

There are only a few studies on this subject at present. I shall try and summarize in what follows a few points on which both experimentation and analysis could usefully dwell in the future.

PSYCHOPEDAGOGY OF THE INTRODUCTION OF A FORMAL LANGUAGE

Here are some of the main points in which the novelty of a FL is more apparent, with respect to a NL, and the probability of finding well defined cognitive obstructions is particularly significant:

Natural languages and formal languages in context

When dealing with the acquisition of a FL, considerations based on an amalgam of processes of *production* (program writing), *comprehension* (program reading) and *interaction* (running a program on a computer, debugging it, transforming it and interpreting its results) are as unsuitable as when applied to an amalgam of writing, reading and speaking/understanding in a NL.

In particular, *writing* a program is very different from *reading* one, especially one written by someone else (Wirth, 1976, p. XV; Wertz, 1981; Arsac, 1984). The local understanding of its lexical, syntactic and semantic structures does not necessarily imply a global understanding of the underlying project. It is clear than a similar difficulty is present in the practice of a NL too; children can read a story word by word and then be unable to tell what the story is about. However, new aspects arise with the introduction of a FL. In particular, the long-range effects of some of its lexical elements on the unfolding of a program (for instance, the assignation of the type and range of the variables and the role of conditional and recursive clauses) make often the program, seen as a whole, particularly opaque. This is only one of the several aspects taken by the difficulty to integrate from local to global in a FL, on which more will be said below (for examples of these difficulties during LOGO acquisition, see Dionet *et al.*, 1985, 1987).

The act of naming

We accept and socially reconstruct the meanings of the words we use in a NL. Proust —in *Du côté de chez Swann*— tells us of his childhood, when he was still near «l'âge où on croit qu'on crée ce qu'on nomme». But the creative act of *naming* is, in our adult world, reserved to a privileged few.

In a FL, on the contrary, *naming* —e.g., enlarging the spectrum of the given initial lexicon of *primitives* by defining *procedures* as new primitives— is considered at present one of the most useful features of a language. (My examples will be taken from programming languages, but the argument would be considerably the same for FLs in general). The names of the new procedures can be arbitrarily assigned (they are defined by what they *do,* once embedded into a program, not by what they *say;* a procedure calle SQUARE can very well, without the computer ever protesting, execute a circle). The feeling of power that this facility creates in children is often manifested by the choice they make for the names of their new procedures: nonsense words, ironical dabs at comrades, four-letter words...

What is more, in a FL the primitives themselves are not as sacred as such, as they too can sometimes be stripped of their initial meaning and given a new, arbitrary one. Syntactic rules can be equally redefined inside the language, even if with some more work and by paying attention to the danger of possible contradictions. It is strange: the very rigid nature of the LF, its capability to lead with no ambiguity to the construction of new meanings starting from its building blocks, lead to this large freedom in the assignation of names and rules. The point here is that it is only the *result* of the procedure that matters, once the procedure is read by the computer, not its *name,* which can be chosen according to personal —and not necessarily social— preference and can be changed at any moment at will. In this sense, and surprisingly, a NL is more rigid than a FL, as it demands *social* consensus before change is accepted.

We should however not forget the opposite aspect that can take the act of naming in the inter-relation NL-FL: the over-generalisation to a FL of a NL lexical term and semantics. A procedure called SQUARE will be often interpreted automatically as producing a square, even if it does not. The primitive STOP in LOGO (which only passively returns control back to the most recent active procedure) will be almost always be interpreted as an indication that the program will halt (Kurland *et al.*, 1985).

In this newly found power of naming there is something probably deeper than what we have discussed above: the fact that here *naming* does indeed imply *creating* and *solving.* To state it in the words of Wertz, 1981: «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».

Procedural vs declarative formal languages

When we define the notion of a *circle* in a NL we have at least two alternatives:

— (declarative) *circle:* the locus of all points equidistant from a given

— (procedural) *circle:* the locus of all points through which we pass if we take a string, fix it to a peg and turn around, without winding it on the peg.

The first alternative provides no suggestion on how to concretely draw a circle, while the second does. The first alternative tells us of a fundamental property of the circle (i.e., the existence of a privileged point, equidistant from all points of the curve), while the second does not. A procedural FL, such as LOGO, can accomodate both definitions, but the procedural one will be more elegantly and efficiently programmed than the declarative one. A declarative FL, such as PROLOG, will more easily accomodate the declarative definition. An interesting debate on the relative merits of procedural vs declarative languages in education is currently going on (see Johanson, 1988; Scherz et al, 1990). On the possible tranfer of competence from one to the other type of FLs, see Shneiderman, 1980, ch.8; Mendelsohn, 1988.

It seems to me, however, that this distinction into two families of FLS is too rigid and perhaps not as compelling at it seems. Let us take an example from a procedural language. In the same way as, in a NL, we can play at will with words and put them together in a well-formed string and see what happens (the production of nonsense being in this case more than probable, but who knows?), so we can play at will with primitives and put them together in a well-formed procedure, named NONSENSE. What will the result of the game be? Hardly nonsense, as the computer will respectfully calculate or graphically represent the end result of NONSENSE. But our procedure will *not* be the procedural representation of a given notion or project or problem, just because we do not know —due to the opacity of the global program structure— *what* the procedure will actually do. The object named by our procedure is thefore, in this case, defined in a *declarative* way: NONSENSE is nothing else than the listing of its corresponding procedure; or, if you prefere, nothing else than the outcome of the procedure, as it will be provided by the computer at each run. Notice that —as it always happens with a procedure— the name of the procedure refers both to the procedure itself and to its effect once embedded into a program.

The temporal effect. The unfolding of a program

There seems to be a qualitative difference in the way an expression in a NL is embedded into, and defines at the same time, an unfolding *time,* with respect to the analogous situation in a FL. Most of the aspects of our experience that we represent into language are *processes,* so that some sort of temporal dimension is intrinsic to them. But, in a FL, the temporal sequence of *actions* —as defined by the sequence of primitives and procedures in the program— is of very little help in comprehending the unfolding of the program.

Procedures and, more in general, programs, are formally *written* as sequences of primitives and therefore of *orders* given to the computer. But, if we *read* them as such, we are far from following the way the program unfolds in time (Arsac, 1984). The significant sequence is not that of *actions* but that of *configurations:* each new action makes the computer go from one configuration to another, and perhaps turn back, and possibly enter a recursive loop, and so on. To the linear sequence of primitives and procedures corresponds, in general, a highly non-linear development of the computing activity.

It is true that in a NL too (in particular, during a reading activity) the sequence of words in a text does not correspond to a sequential apprehension of the text meaning: the mind runs back and forth and reconstructs, reinterprets, reintegrates the accumulated information at each step. But there are no examples of loop structures in NLs (but for vicious circles in argumentation), and those examples, as are given, of recursive structures are at best doubtful (Vitale, 1990a). We would be tempted to say that the way *time* is implicitly present in both NLs and FLs is qualitatively different and that this difference can justify much of the difficulties encountered in the mastering of a FL, in particular its recursive potentialities. Our everyday time is complex and confusing enough, but at least it does not seem to present the manifold, branching structure that characterizes the time of informatics.

When we speak, to whom do we speak?

When we speak in a NL, in a situation of discourse, we are continuously —if often unconsciously— adjusting our arguments to the reactions we face. Even more, we are generally able to *repair* an unwilling slight to our interlocutor by adding a few words that will help restructuring the whole argument as to avoid any possible misunderstanding. When we write a program in a FL, to whom do we speak? There is communication to the computer, of course; but, is there *discourse*?

It is a well known fact to teachers that pupils and students do not, in general, even try to read and interpret the error messages sent by the computer (in a clumsy language that is however much richer than the corresponding programming language. In LOGO, for instance, the asking for HAT —if this term does not correspond to either a primitive or a defined procedure— provokes the message on the screen: I DO NOT KNOW WHAT TO DO WITH HAT, where most of the terms used are *not* LOGO primitives).

This indifference to error messages can be partly due to the clumsiness of the language; but I think that it has mostly to do with the difficulty to know, or to represent clearly to ourselves, which is our interlocutor when programming in a FL: who is listening to us and to whom should we be listening to.

In LOGO, an additional difficulty presents itself (see, in particular, Hofmann et al, 1987): that of discriminating between the computer and the turtle. When we give the order FORWARD 50, it is easy to represent the order as given to the turtle, that is so asked to step forward 50 turtlesteps. But if we ask EQUAL? 35, it will not be the turtle that will perform the necessary arithmetics to give us an answer. The situation becomes more complex, and correspondingly the mastery of our FL less secure, if we want to take into account the possibility that the computer (or the turtle?) could also ask *us* questions during the run (for instance, by READCHAR); and that it could print messages that are not in its programming language but in our NL! A program can therefore become a hybrid of a FL and some more or less NL (outputs of error messages and information that we require the computer to give us during the run).

As we have seen, it is hard to convince ourselves that, while programming in a FL, we are in some sort of discourse situation. The difficulty is to identify the interlocutor and to learn how to listen to it, instead of giving only orders to it. A good understanding and a transparent representation of the *functional architecture* of the computer could help here (Pylyshyn, 1984). Again, we are led to the proposal that the computer —in its functional structure, if not in its physical realisation— be seriously taken by both pupils and teachers as an interesting and sometimes surprising object of knowledge.

Modelling in a FL

There is a double modelling activity going on hile applying a FL to the representation and solution of a problem.

The first one leads from our experience of a given causal or formal situation to the identification of its underlying logico-mathematical components and to the choice of a suitable representation; this can be done in several modalities —verbal, gestual, graphical...— and can possibly (but not necessarily) be formalized in the language of mathematics. Some of the richness of the physical situation has been lost here, but what has been gained is a better definition of the limits and scopes of the model.

The second one leads from the chosen representation to its description in a FL, which implies —to use again the words of Wertz— «The formalisation of a problem *and* its solution». A formalisation that has now to take into account the specificity and the possible idiosyncracies of the given FL; a solution that has to take into account the specificity and the possible limitations of the given computer system.

Neither FL nor computer are neutral with respect to our choices.

Here again we would insist that this transcription be itself considered by pupils and teachers as an object of knowledge. There are generally several paths available, equivalent in the final numerical result (apart from small discrepancies due to different approximations) but often very different in the cognitive regions they pass through and explore. The same FL can permit very different transcriptions of the same model; different FLs will permit an even larger spectrum of transcriptions. To follow in detail at least some of these paths, by comparing them and by understanding their rationale, can help deepening our mastery of the space of programming.

Should formal languages imitate natural languages?

In the Sixties, there was a tendency to ask that FLs become as much as possible similar to a NL (that, of course, would be English); but, as Shneiderman notes (p. 199), «Reading and comprehending an English-like programming language is relatively easy, but writing syntactically correct code is a challenge. The closeness of ENGLISH to English makes it difficult to remember the grammar of ENGLISH: an elegant demonstration of *proactive interference*, the confusion between what you know and what you are trying to learn».

It seems now that this trend has lost its drive and that both common users and professional programmers agree that it is better to have a specific, limited scope FL —with its advantages of rigour and disadvantages of rigidity— for each domain of programming, instead of dreaming either an English-like FL or a universal (esperanto) FL. I think too that it is better to leave the two realms apart: NLs, with their helpless richness and their capability of subtle adapting to every new situation or context; FLs, with their rigid structures and specific adaptation to specific situations. A «poor» FL (by NL standard) can be very rich and suitable for exploration in its well defined domain (for instance, in the construction of LOGO-microworlds). It can therefore help more in concentrating on the cognitive and representative aspects of a given task than a «richer» but less well defined FL.

CONCLUSIONS

What I am advocating, in this short presentation of the psycho-cognitive and psycho-pedagogical aspects that dominate the first encounter of pupils with computers in the school context, is the need to be sensitive to the way the pupil/computer relation evolves as a consequence of the interest and of the difficulties, inherent to the introduction of a formal language as a necessary interface between the actor (the pupil) and the machine (the computer). Lexical, syntactical and more technical aspects are equally important, but it seems to me that a satisfactory integration of elementary programming into school curriculum will not be obtained, unless the *language and communication problems* briefly outlined in the present paper are properly taken into account *in the educational context.*

References

ARSAC, J. (1984). La conception des programmes. *Pour la Science, 85*, 1984, 105-113.

DIONNET, S.; MARTI, E.; VITALE, B., and WELLS, A. (1985). Représentation et contrôle global-local du mouvement chez l'enfant dans la programmation LOGO. *Revue Française de Pédagogie*, 72, 13-23.

DIONNET, S.; GUYON, J.; and VITALE, B. (1987). Aspects psycho-cognitifs de l'activité informatique, in M. Descolonges and F. Enel (Eds.): *Evaluation des clubs d'informatique en France; analyse psycho-sociale et psychocognitive.* Rapport au Ministère de la jeunesse et des sports et au Ministère de la culture. Paris.

HOFMANN, B.; MARCELLUS, O.; REY, F.; and VITALE, B. (1987). TATUE ou la relation élève-ordinateur; Une évaluation formative du cours d'informatique au Cycle d'orientation. Genève: DIP-CRPP.

HOFMANN, B.; DE MARCELLUS, O.; REY, F.; and VITALE, B. (1989). Observation de cours d'initiation à l'informatique; Complément à la recherche «TATUE»,. Genève: DIP-CRPP.

JOHANSON, R. P. (1988). Computers, cognition and curriculum; Retrospect and prospect. *Journal of Educational Computing Research, 4*, 1-30.

KRENDL, K. A. and LIEBERMAN, D. A. (1988). Computers and learning; A review of recent research. *Journal of Educational Computing Research, 4*, 367-389.

KURLAND, D. M. and PEA, R. D. (1985). Children's mental models of recursive LOGO programs. *Journal of Educational Computing Research, 1*, 235-243.

MENDELSOHN, P. (1988). Les activités des programmation chez l'enfant; Le point de vue de la psychologie cognitive. *Technique et Science Informatique, 7*, 47-58.

PEA, R. D.; and KURLAND, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*, 137-168 (also in R. D. Pea and Sheingold (Eds.): *Mirrors, of mind; Patterns of experience in educational computing.* Norwood: Ablex, 1987, 147-177).

PYLYSHYN, Z. W. (1984) *Computation and cognition; Toward a foundation for cognitive science.* Cambridge (Ma): MIT Press.

SCHERZ, Z.; GOLDBERG, D. and FUND, Z. (1990): Cognitive implications of learning PROLOG; Mistakes and misconceptions. *J. Educat. Comp. Res.*, *6*, 89-110.

SHNEIDERMAN, B. (1980). *Software psychology; Human factors in computer and information systems.* Cambridge (Ma): Winthrop.

VITALE, B. (1988). Epistemología y pedagogía de la introducción de los niños a la informática. En M. Aguirregabiria (ed.): *Tecnología y educación,* Madrid: Narcea 1988, 138-148. (Epistemology and pedagogy of children's approach to informatics, *International Conference on Education,* Bilbao 1987).

VITALE, B. (1989). Relation entre théorie et pratique; Le cas de l'informatique. *Education et Recherche, 11,* (3), 23-34.

VITALE, B. (1990). L'intégration de l'informatique à la pratique pédagogique; I: Considérations générales pour une approche transdisciplinaire. Genève: DIP-CRPP.

VITALE, B. (1990a). Elusive recursion; A trip in recursive land. *New Ideas in Psychology, 7,* no. 3.

WEIZENBAUM, J. (1984). *Kurs auf den Eisberg; oder «Nur das Wunder wird uns retten», sagt der Computerexperte.* Zürich, Pendo.

WERTZ, H. (1981). Some ideas on the educational use of computers. *Proceedings of the Annual Conference of the A.C.M.*, Los Angeles, 101-107.

WIRTH, N. (1976). *Algorithms + data structure = programs.* Englewood Cliffs: Prentice-Hall.