

# Contributing to the Development of Linguistic and Logical Abilities through Robotics

**G. Barbara Demo**, *barbara@di.unito.it*  
Dept of Informatics, Turin University – Italy

**Giovanni Marcianó**, *marciano@irrepiemonte.it – margi@bmm.it*  
National Agency for Development of Schools - Piedmont, Ministry of Public Education - Italy

## Abstract

In primary schools robot programming is fun and may therefore represent an excellent tool both for introducing to ICT and for helping the development of logical and linguistic abilities of schoolchildren. Core of our project is NQCBaby, a Logo-like robot programming language, which – in the tradition of Logo – is child-oriented rather than robot-oriented like NQC. In the early 90's S. Papert had already suggested the educational use of "small robots programmed in Logo" [Papert 1993]. We had robots, of course we had Logo, but we had no Logo for robots. NQCBaby is an Italian version of a Logo-like interface to NQC, so that children used to deal with the Turtle can transfer and adapt their implicit abilities to robots, also discussing the differences. In the kind of activities we consider, schoolchildren start by using a very simple language, the first level of NQCBaby, to interact with the simplest robots RCX; later, as their robot-assembly experience grows, they move on to successively richer versions of the language. As a matter of fact, the constructive pedagogical methodology (and consequently our tools) structures the learning of NQCBaby in several steps, starting with NQCBaby01 up to NQCBaby05, as new hardware components are introduced to build new more sophisticated kinds of robots. In the meanwhile children are also introduced to NQC, the "real" robot language, by looking at how their descriptions of robot behaviors are translated into NQC "in order to be understood by robots". This step-by-step activity of schoolchildren is coordinated with the parallel learning of the basic linguistic abilities in their native language. A software environment, based on a precompiler from NQCBaby to NQC, is currently being developed for supporting the project principles, in a sequence of levels of increasing complexity and abstraction from NQCBaby01 to NQCBaby05.

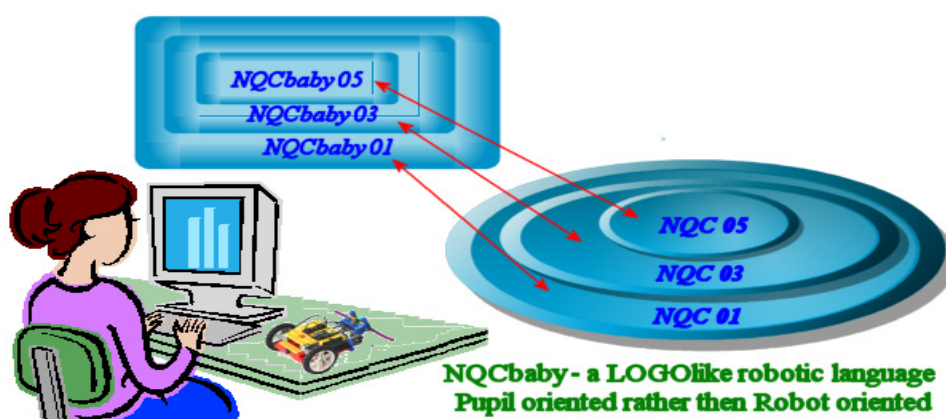


Figure 1. Children assemble a robot, decide behaviour, describe it in NQCBabyh, and look at the resulting specification in NQC; then they enter the next step to learn the next level NQCBabyh+1

## Keywords

Constructive learning, robotics, Logo, precompiler, RCX, NQC

## 1 Introduction

In primary schools robot programming is fun and may therefore represent an excellent tool for both introducing to ICT and helping the development of logical and linguistic abilities of children in their first years of instruction. Robotics experiences have been carried out in Italian schools since 2000-2001 when the very first project “SeT: Let us build a robot together” was proposed; its description can be found at <http://www5.indire.it:8080/set/microrobotica/>. These activities with children convinced us that teaching digital technologies in primary schools should be concerned with the language used for interacting with robots rather than be limited to making robots move as we want to. The project here described introduces children to programming the small Mindstorms RCX robot through different linguistic steps where the programming language evolves with the knowledge the pupils acquire of the robot components. When pupils, at the beginning of their robot experience, build the simplest robots they also use a simple LOGO-like language, called NQCBaby, to interact with them. As children knowledge evolves also the interaction with robots evolves: pupils learn that not only new words but also new way of structuring sentences are needed when different robot components are introduced and different behaviours may be designed and specified.

Thus, for primary school pupils learning robots programming also becomes an opportunity to develop their linguistic and logical capabilities within a context where pedagogical aspects are focused rather than, or before, introducing technological terminology. Also, children can, at the same time, use similar LOGO languages to make turtles move in virtual environments (or microworlds) and to manipulate LEGO RCX robots which are real objects.

At the beginning of the project, we considered using the textual language NQC, with its Bricx Command Center (BCC) integrated development environment, rather than iconic languages because the use of the same format allows most immediate influence exchange between linguistic competences pupils are collecting in their native language and during the robot programming activities [BricxCC, Marcianó 2005]. Also native language translations of most used robot programming languages were considered because they are obviously nearer to the pupils. But analyzed proposals resulted unsatisfactory since we consider mandatory that languages are designed to be pupil oriented rather than robot oriented, and, for example, concepts from the child world must be used to describe a robot behavior. Already in [Marcianò 2004] we pointed out that LEGO system, available from 2002 with Mindstorm (RCX) kits, is too far from the LOGO philosophy because affected by a technological approach. Similarly, from the first experiences of programming the RCX robot with NQC, using the Bricx Command Center (BCC) integrated development environment, we were convinced that pedagogical aspects were sacrificed to technological components.

The LOGO-like language presented by Marcianò in [Marcianó 2006a] is specifically developed for pupils using Lego RCX based kits. In fact, the present paper collects the outcomes resulting from several experiences based on the theoretical concepts inspiring that paper.

NQCBaby, here described, is a proposal toward a language more pupils oriented. During their first experiences, pupils are given robot-assembling boxes containing different sets of RCX components. Of course pupils build, under the guide of the teacher, different robots with boxes having different sets of components, usually robots with an increasing number of abilities as the number of components increases. As an example given a light sensor it obviously becomes possible to distinguish different levels of lights thus becomes possible for a robot to follow a path on the floor marked by a black ribbon. In our approach the language evolves with the pupils knowledge of the environment. The following Section 2. describes the language NQCBaby by means of different steps as it is introduced to the children in our experiences together. Examples of robot programs using different levels of the NQCBaby language are given in Section 3. After the language has been introduced we come back to motivations why we consider it more suitable to children (Section 4).

In Section 5 a short description is given of the software solutions we considered to integrate NQCBaby with NQC and then with Bricx Command Center until the precompiler-postcompiler solution of the language currently under development.

## 2 NQCBaby by steps

In this section different robots are considered with language elements needed to direct them. During the first meeting each pupils' group is given a box only containing components needed to assemble the simplest robot: it is the so called *chariot* having two wheels moved by motors A and C and no sensors neither lights nor pincers. To interact with this simple robot a simple language is needed, shown in the left column of Table 1. Each interaction begins by saying hallo to the robot called by name, Robby, and finishes saying ciao (bye). The translation of each NQCBaby instruction into the corresponding NQC sequence of instructions is shown in the right column of Tables that follow.

Robby >	task main() {
avanti(v) >	OnFwd(OUT_A+OUT_C); Wait(v);
indietro(w) >	OnRev(OUT_A+OUT_C); Wait(w);
destra(t) >	OnFwd(OUT_A);OnRev(OUT_C); Wait(t);
sinistra(u)>	OnFwd(OUT_C);OnRev(OUT_A);Wait(u);
veloce(x) >	SetPower(OUT_A+OUT_C, x);
avantisempre>	while(true){OnFwd(OUT_A+OUT_C); };
ripetisempre>	while(true) {
ripeti(z) >	repeat (z) {
fine >	Off(OUT_A+OUT_C); }
ciao>	Off(OUT_A+OUT_C); }

Table 1. NQCBaby01 Language.

Examples of different behaviours specifications using the above primitives are the following (from // the translation of Italian instructions is given):

1. Robby  
avanti(100) indietro(100) // forward(100), back(100)  
ciao
2. Robby  
ripeti(4) avanti(100) destra(90) fine // repeat(4) forward(100), right(90) end  
ciao

Behaviour 1. is one of the first try a pupil makes for testing if his/her robot moves forward for a while (avanti(100)) and then comes back to its starting point (indietro(100)).

Behaviour 2. is soon produced by children who already have used some Logo, which is often the case in schools concerned with their pupil's introduction to technology. Children try to give the robot the same sequences of commands already used with the Logo turtle for example the one in 2. is for designing a square. Obviously from this code sequence it turns out that right(90) does not mean to the robot to turn right 90 degrees as to the turtle. Discussions after this try are quite valuable for learning differences between the two environments.

Code sequence 2. also shows the use of the primitive fine (end) to terminate a repeat sequence of commands rather than having the sequence between {} brackets not present on an Italian keyboard.

As a second step, we expand NQCBaby01 with instructions shown in Table 2 concerning light and touch sensors. Usually the touch sensor is already introduced during the second meeting with a class. One new word (sensor\_touch) appears in the language for the new component and some other words appear to say what to do with the new component: se\_tocca i.e. if\_touches.

sensore1_tocco >	SetSensor(SENSOR_1, SENSOR_TOUCH);
se_tocca >	if (SENSOR_1 == 1) {
Accendiluce >	On(OUT_B);
Spegniluce >	Off(OUT_B);
aspetta(z) >	wait(z);

Table 2. Primitives for NQCBaby02

Accendiluce and Sspegniluce are primitives for switching on and off the lamp on B actuator. Having motors on A and C, as decided above, the lamp goes on the actuator B. Expressions in Table 2 allow to write programs like the following where the robot goes forward until something is hit. Such hitting causes switching on the light, moving to the right and switching off the light. Again the translation follows the //.

```

Robby
sensore1_tocco      // touch sensor goes to sensor number 1 of the robot
  Ripetisempre      // repeat always
    avanti(10)      // forward(100)
    se_tocca        // if hits
      accendiluce   // switch on the light
      indietro(100) // back(100)
      destra(50)    // right(50)
      spegniluce    // switch off the light
    fine            // end
  fine
ciao

```

In the third step we introduce a second touch sensor to control the rear of the robot (where sensor number 2 is located). When we bring out of the robot kit the light sensor, we have to settle the value we consider the border value for light (chiaro) and dark (scuro). In our experiences it is reasonable to have 48 as this border value. Light sensor goes on robot sensor position number 3 that is under the chariot on the front.

Expressions in Table 3 are added to the language:

sensore2_tocco >	SetSensor(SENSOR_2, SENSOR_TOUCH);
se_toccadietro >	if (SENSOR_2 == 1) {
senti3_luce	SetSensor(SENSOR_3, SENSOR_LIGHT);
se_chiaro	if (SENSOR_3 > 48) {
se_scuro	if (SENSOR_3 <= 48) {

Table 3. Primitives for NQCBaby03

On a fourth step, the language is enriched with primitives in Table 4, that provide new controls directed to the light sensor and for the execution of an alternative sequence of commands, by using the else component of the language

sinoache_chiaro	until (SENSOR_3 > 48);
sinoache_scuro	until (SENSOR_3 < 48);

altrimenti	else
acaso(n)	Random(n)

Table 4. Primitives for NQCBaby04

Notice the introduction of the powerful and fascinating primitive *acaso(n)* (*Random(n)*) to explore the randomness world.

On the final step, instructions to control two procedures named *prima* (*first*) and *seconda* (*second*) are introduced as shown in Table 5.

uno	task prima() {
fai_uno	start prima;
stop_uno	stop prima;
due	task seconda() {
fai_due	start seconda;
stop_due	stop seconda;
fine_fai	}

Table 5. Primitives for NQCBaby05

### 3 Some examples of robots programs using the above primitives

Some examples of robots programs are shown using the above primitives.

```

Robby
veloce(3) avanti(100) veloce(7) indietro(100) // speed(3), forward(100), speed(7), back(100)
ripeti(3) destra(90) sinistra(90) fine // repeat(3) right(90) left(90) end
ripeti(2) indietro(10) avanti(20) fine // repeat(2) back(10) forward(20) end
ciao // hallo
    
```

A pupil in a second class of a primary school, seven years old, has developed this very simple program based on **NQCBaby01**.

```

Robby
senti1_tocco
ripetisempre
    avanti(10)
    se_tocca
        indietro (100) destra(100)
    fine
fine
ciao
    
```

During a stage a group of teachers has developed the above program based on **NQCbaby02** defining a robot that goes around the classroom, avoiding obstacles.


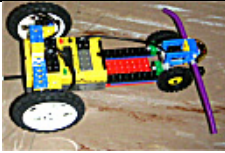
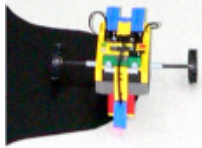
```

Robby sensore3_luce
  ripetisempre
    se_scufo avanti(10) accendiluce destra(10) spegniluce fine
    avanti(10) sinistra(10)
  fine
ciao
  
```

A group of pupils in their fourth year of a primary school (nine years old) has developed the above program based on **NQCBaby03**. It defines a robot able to go forward along a border between a black (on its left side) and a white area (at its right side).

## 4 Why NQCBaby is useful

While introduced to NQCBaby as described above, pupils in first years of primary schools progress in the development of their linguistic and logical abilities because of the use of their native language but also, and more relevant, because they use a **higher level language** wrt NQC language. Higher level because of its *compactness* and because of the *use of children well known concepts rather than using the names of robot components*. As for compactness, we can see from the Tables in the previous Section that several NQCBaby commands correspond to more than one command in NQC. In NQCBaby *concepts well known* to pupils are used such as dietro (on-the-back) for the primitive se\_toccadietro (if touches on the back) to distinguish it from the simple se\_tocca (if touches) used for if (SENSOR\_1 == 1){, introduced in the language when only one sensor\_touch was given to the pupils while building the robot. If, on the one side, we lose in flexibility because, as an example, in NQCBaby sensor-2 is devoted to a touch sensor and sensor-3 is devoted to a light sensor, on the other side in their first steps children find more natural describing robot behaviour in NQCBaby. They are more involved in the activities and develop a lot of different programs for their small robots during a single lesson, comparing their jobs and improving their programs.

Robby ciao veloce(x) avantisempre avanti(v) indietro(w) destra(t) sinistra(u) ripeti(z) ripetisempre fine		NQCBaby01 “ “ “ “ “ “ “ “ “				
accendiluce spegniluce aspetta(z) sensore1_tocco se_tocca			NQCBaby02 “ “ “			
sensore2_tocco se_toccadietro sensore3_luce se_chiaro se_scufo				NQCBaby03 “ “ “ “		
sinoache_chiaro sinoache_scufo altrimenti					NQCBaby04 “ “	

acaso(n)		“		
uno			NQCBaby05	
fai_uno			“	
stop_uno			“	
due			“	
fai_due			“	
stop_due			“	
fine_fai			“	

Table 6 – Comprehensive view of NQCBaby language

Also, syntactical components as { } , ; from NQC are not present in NQCBaby: the children learn they exists when they begin looking at their code translation in NQC and are given motivations for having them when they ask. After NQCBaby05 they will use NQC for describing their robot behaviour, from the beginning and with the whole expressiveness of the NQC language: but, at that point, they will be familiar with many robot-only language components.

In Table 6 commands of NQCBaby language are listed all together in the first column. From this inclusive view one can see how NQCBaby is in many instructions syntactically near to Papert's Logo which is indeed our constant reference. Logo is popular in Italian primary schools and a number of teachers have developed activities syntonic with Logo philosophy [Papert99]. When beginning an experience of robot programming in a new environment teachers have reported of quite different reactions of pupils depending on whether NQCBaby or NQC are proposed to begin with. Pupils are obviously more confident finding commands having the same names of Logo commands though commands are different because, as an example, for robot we have speed and time while for turtles we have distances and degrees as parameters in some instructions.

Moreover, conceived for third (or even second) year school children which are 7 years old, NQCBaby goal is to encourage pupils to play with concepts such as avanti (forward) / indietro (backwards), sinistra (left) / destra (right) related to the robot position. Also playing with time, space, speed to control robot movements, allows children to manipulate complex concepts that cannot be explored in a different way.

After this “manipulation step”, children are required to program their robots for executing specific tasks. Through these experiences they learn to be precise, because if they do not follow language rules their programs are not understood, also they learn precision in a pleasant and constructive way (the robots do not behave as they would like to). This is common to learning how to program in general but it is a good result when teachers find it out in their first years class, after some programming experiences, having seen actively involved most of their pupils. On this concern Simonetta Siega, teacher in the primary school of Baveno, presents her experiences in the workshop titled “**Seven years old pupils programming turtles and robots using Logo-like language: NQCbaby**” While experimenting, using the first macro version of NQCBaby dealt with in next Section, we had several hints to improve the language toward being more-child-oriented directly from pupils!

## 5 From NQCBaby to NQC: how

Proposed beginning 2005 in a very simple form to verify its usefulness we are currently devoting to NQCBaby a typically technological activity of implementing a precompiler converting NQCBaby texts into NQC programs.

### Macro defined language: quick but not coherent

At the very beginning NQCBaby has been proposed by Marciano as a macro defined language [Marciano2005] where every primitive *P* having a NQC-translation *Q* was defined in a line of code as follows:

```
#define P Q
```

This is of course a very simple yet powerful technique for experimenting the different language layers on top of NQC. Obviously it does not support coherence of the answers from the compiler to the pupils. Indeed syntactical errors are signalled on NQC primitives that pupils, at least at the beginning, do not know. Thus our current effort is to implement 1) a precompiler to signal NQCBaby syntax errors on NQCBaby primitives and 2) a postcompiler to translate NQC compiler errors messages on NQCBaby error messages.

The precompiler output can be shown to the children so they can learn what their code looks like in NQC because, as a last step shown in Figure 1., children program in NQC in order to have the entire flexibility of this language in dealing with any assembly of a robot they can envisage: even those that children call “crazy”.

### NQCBaby Precompiler

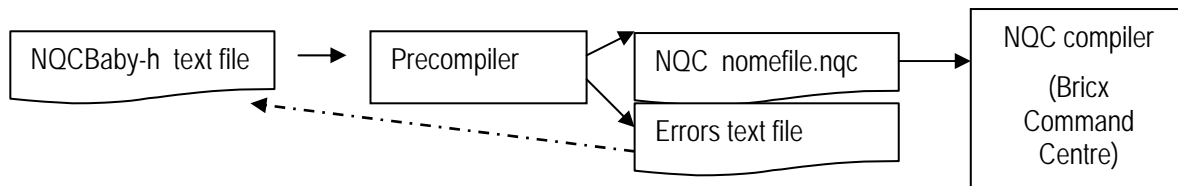


Figure 2. Children assemble a robot, compose its behaviour in a NQCBaby-h text file, go back to the beginning if errors are found or look at the translation in a NQC file to be compiled and sent to the robot.

In this paragraph we briefly describe how NQCBaby precompiler does currently work, as shown in Figure 2. Children define the behaviour of their robot according to an NQCBaby any level language in a text file. This is given in input to the precompiler which either returns as a result a syntax error notice, obviously concerning the NQCBaby code, or returns a text file containing an NQC code sequence as from Tables above. The latter is then given in input to the NQC compiler in BrixCC for the translation into an executable code sequence.

Our present activity is toward implementing a tool easier to use for children and looking similar to the Bricx Command Center where they work for the final translation from NQC and for sending the executable code to the robot.

Indeed the tool under development will have two work modes. First use the mode NQCBaby-only mode, where only NQCBaby is considered consistently with the environment provided by the BCC (Bricx Command Center) environment. Thus an editor of NQCBaby supports syntax highlighting and code completion for the NQCBaby language.

The second work-mode is called NQCmode because it adds to the NQCBaby only mode the translation of each NQCBaby primitive into NQC commands. Thus pupils can see does it look like the other language the robot can understand: it is a first step toward learning the NQC language.

The postcompiler integrates with the precompiler to rewrite errors found in NQC code by NQC compiler, into syntax errors in NQCBaby code listed at the end of the NQCBaby code page. Clicking on the error message BCC (Bricx Command Center) highlights the line where the error has been found: a professional function realized in a friendly way.

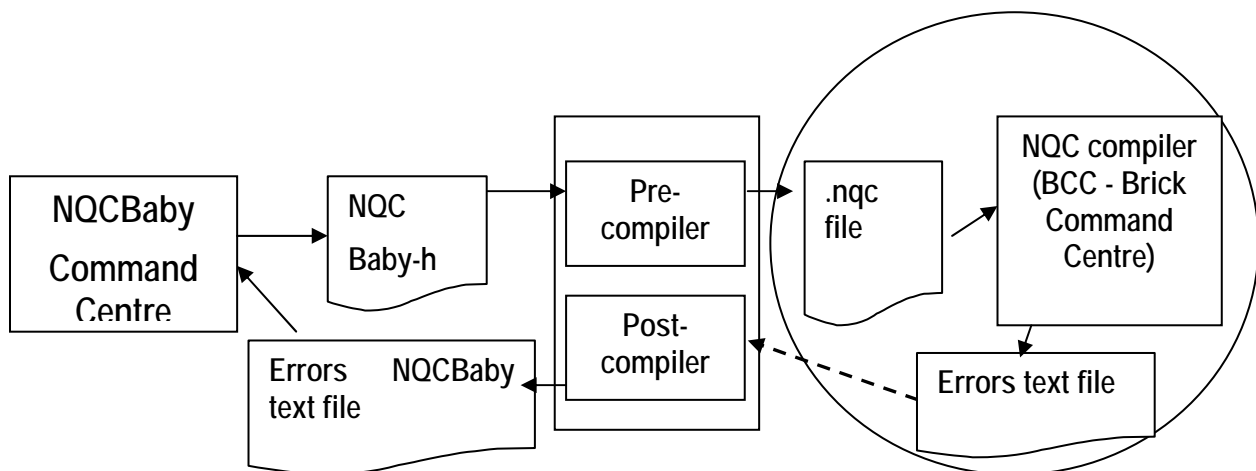


Figure 3. The new component Postcompiler with errors listing on NQCBaby.

### NQCBaby Postcompiler

Next development in our systems is shown in Figure 3. where the Postcompiler component is the relevant new entry.

Errors found by the BCC (Bricx Command Center) during compiling NQC into the executable code are captured and translated by the Postcompiler, using some Precompiler data, into NQCBaby errors, Thus the Postcompiler component allows the translation of errors found in NQC code by NQC compiler, into syntax errors in NQCBaby code. Errors are then listed at the end of the NQCBaby code page. Thus when used in NQCBaby-only mode the system will coherently signals errors on the NQCBaby code: essential for pupils during their first experiences of robots programming when they don't yet use NQC.

## 6 Final remarks and acknowledgments

Proposed for five years, the ideas described in this paper have been tested by many colleagues and their pupils. Simonetta Siega and her colleagues deserve a special thank-you for suggestions and comments on children reactions during their several years of robot experiences in primary schools of Baveno, Tortona and other schools in Piedmont. Images and reports with comments on these activities can be seen on the web at: <http://robotica.irrepiemonte.it/robotica/index.htm>

Our goal with NQCBaby is to develop a methodology rather than a language. Indeed, our next activity will concern moving the project to languages for programming other small robots: we think this can be done adding to the methodology some other components. Currently we are working on defining an analogous by-step language and set of tools for LEGO NXT programming (NBC language referred). With NXT a number of different robots can be built: thus we are defining a language where robot programs contain two distinct components: the robot *configuration description* and the robot *behaviour description, that is the program*. This is an evolution in programming of small robots that makes it converge toward the fundamentals of computer programming defined as *Algorithms+ DataStructures = Programs* by Wirth in his work about Pascal around the same years when Papert and his group were working on Logo [Wirth 1971, Wirth 1976, Papert 1980].

## References

BCC - *Bricx Command Center site*, at: <http://bricxcc.sourceforge.net/>

Marcianó, G. (2004) *Linguaggi robotici a scuola*, at

[http://margi.bmm.it/old/robotica/Linguaggi\\_robotici\\_a\\_scuola.pdf](http://margi.bmm.it/old/robotica/Linguaggi_robotici_a_scuola.pdf)

Marcianó, G. and Siega, S. (2005) *Linguaggi robotici per la scuola*. In Proceedings of DIDAMATICA 2005, Edited by A. Andronico, N. Cavallo, A. De Michele, M. Fasano, Potenza, pp. 185 - 196.

Marcianó, G. (2006a) *Informatica come Linguaggio*. In Proceedings of DIDAMATICA 2006, Edited by A. Andronico, F. Aymerich, G. Fenu, Cagliari, May 2006. pp. 185 - 196.

Marcianó, G. (2006b) *Robotica project site*, at

[http://robotica.irrepiemonte.it/robotica/linguaggi/doc/linguaggi\\_robotici\\_scuola\\_2006x.pdf](http://robotica.irrepiemonte.it/robotica/linguaggi/doc/linguaggi_robotici_scuola_2006x.pdf)

Miglino, O.- Lund H. H. – Cardaci, M. (1999) *Robotics as an Educational Tool* in JI. of Interactive Learning Research (1999) 10(1), 25-47

Papert S. (1980), *Mindstorms: Children Computers and Powerful Ideas*, Basic Books, 1980.

Papert, S. (1993). *The children's machine: rethinking schools in the age of the computer*, Basic Books, 1993.

Papert S. (1999), *Logo Philosophy and Implementation*, LCSl, Canada, 1999

Wirth, N (1971) *The Programming Language Pascal*. Acta Informatica, 1, (Jun 1971) pp. 35-63

Wirth, N. (1976) *Algorithms+Data structures=Programs*, Prentice-Hall Series in Automatic Computing. Eaglewood, N.J.: Prentice-Hall Inc. 1976.