The construction of knowledge about programs

Federico Gómez
Instituto de Computación -
Facultad de Ingeniería
Universidad de la República
fgfrois@fing.edu.uy

Sylvia da Rosa
Instituto de Computación
Facultad de Ingeniería
Universidad de la República
darosa@fing.edu.uy

Abstract

This paper presents an empirical study that uses the linear search problem to investigate the process of construction of knowledge about programs.

The study was carried out in an introductory programming course of a Computer Engineering Undergraduate Program, using two imperative programming languages (*Pascal* and *C++*) in two different groups of students.

Building on the theory of Genetic Epistemology of Jean Piaget and his understanding of the construction of scientific knowledge, the study applies Piaget's principles of a triad of *intra*, *inter* and *trans* stages, and the general law of cognition, in order to analyse the passage from the *intra* stage into the *inter* and *trans* stages. The passage involves the actions students undertake as they design an algorithm to solve the proposed problem, and the reflections needed for them to implement and execute a program on a computer.

The study describes several steps in which students carried out different activities. In order to encourage them to conceptualise and formalise their solutions, individual interviews were conducted with thirteen students, of approximately an hour each. The paper includes selected excerpts of students' responses, our analysis of the results, and some conclusions and future work.

1. Introduction

The study presented in this paper applies principles of Jean Piaget's theory of Genetic Epistemology and his understanding of the construction of scientific knowledge, briefly described in Section 2. The study starts from the resolution of a concrete instance of the linear search problem in which the participants are asked to look for a specific number in a row of numbered cards (*instrumental* knowledge). Subsequently, clinical interviews (similar to those carried out by Piaget) are conducted to help students to explain, in natural language, the method used and the reasons for success in solving the problem (*conceptual* knowledge), turning back to actions if necessary. The study ends with the writing, compilation and execution of a program that searches for a given value in an array of integers (linear search general problem), having to also explain how and why it works when executed on a computer (*formal* knowledge), both at the level of the program's source code (*the textual part* of the program) and its execution on a machine (*the executable part* of the program). According to the theory, in this dialectic interaction between the stages, the students construct conceptual and formal knowledge about the program from their instrumental knowledge of the problem instance (see Section 2).

Multiple investigations were carried out over many years of research by the Computer Science Education Research group of the InCo (Computing Institute, Faculty of Engineering, University of the Republic, Uruguay). The study presented here takes results from previous investigations in which the construction of knowledge about algorithms and data structures was analysed (da Rosa, 2010; da Rosa & Chmiel, 2012; da Rosa, 2015). The contribution of colleagues from Philosophy of Computer Science in the definition of the dual nature of programs as texts and as executable objects was of great importance in the research about the construction of knowledge about programs (da Rosa, 2016; da Rosa,S. & Chmiel,A. & Gómez, F., 2016). In line with this notion, a thorough understanding of the concept of program implies the construction of knowledge on two aspects: the symbolic part of the program (*textual* part) and the physical part (*executable* part) and of the relationship between them (da Rosa & Aguirre, 2018; da Rosa, 2018). The main goal of this study is to investigate the construction of knowledge about programs as dual objects.

The rest of this paper is organised as follows: in Section 2, Piaget's theory is introduced as a theoretical framework. Section 3 presents the design of the empirical study on linear search. Section 4 describes its implementation and includes excerpts from interviews with its participants. Finally, in Section 5, the conclusions of the work are presented and some lines of future work are proposed.

2. Theoretical framework

Piaget's theory offers a model for explaining the construction of knowledge that can be used in all domains and at all levels of development. The central points of Piaget's theory - Genetic Epistemology - have been to study the construction of knowledge as a process and to explain how the transition is made from a lower level of knowledge to a level that is judged to be higher (Piaget, 1977).

The supporting information comes mainly from two sources: first, from empirical studies of the construction of knowledge by subjects from birth to adolescence (giving rise to Piaget's genetic psychology), (Gréco, Matalon, Inhelder, & Piaget, 1963; Piaget, 1978, 1975, 1964, 1974), and second, from a critical analysis of the history of sciences, elaborated by Piaget and Garcia to investigate the origin and development of scientific ideas, concepts and theories. In (Piaget & Garcia, 1980) the authors present a synthesis of Piaget's epistemological theory and a new perspective on his explanations about knowledge construction.

The main idea of their synthesis consists in establishing certain parallels between general mechanisms leading from one form of knowledge to another - both in psycho-genesis and in the historical evolution of ideas and theories - where the most important notion of these mechanisms is the triad of stages, called by the authors the *intra*, *inter* and *trans* stages. The triad explains the process of knowledge construction by means of the passage from a first stage focused on isolated objects or elements (*intra* stage), to another that takes into account the relationships between objects and their transformations (*inter* stage), leading to the construction of a "système d'ensemble", that is, general structures involving both generalised elements and their transformations (*trans* stage), integrating the constructions of the previous stages as particular cases.

2.1. The general law of cognition

In Piaget's theory human knowledge is considered essentially active, that is, knowing means acting on objects and reality, and constructing a system of transformations that can be carried out on or with them (Piaget, 1977). The more general problem of the whole epistemic development lies in determining the role of experience and operational structures of the individual in the development of knowledge, and in examining the instruments by which knowledge has been constructed before their formalisation. This problem was deeply studied by Piaget in his experiments about genetic psychology. From these, he formulated a general law of cognition (Piaget, 1964, 1974) governing the relationship between know-how and conceptualisation, generated in the interaction between the subject and the objects that she/he has to deal with to solve problems or perform tasks. It is a dialectic relationship, in which sometimes the action guides the thought, and sometimes the thought guides the actions.

Piaget represented the general law of cognition by the following diagram:

$$C \leftarrow P \rightarrow C'$$

where P represents the periphery, that is to say, the more immediate and exterior reaction of the subject confronting the objects to solve a problem or perform a task. This reaction is associated to pursuing a goal and achieving results, without awareness neither of the actions nor of the reasons for success or failure. The arrows represent the internal mechanism of the thinking process, by which the subject becomes aware of the coordination of her/his actions (C in the diagram), the changes that these impose to objects, as well as of their intrinsic properties (C' in the diagram). The process of the grasp of consciousness described by this law constitutes a first step towards the construction of concepts.

In previous investigations (as cited above) the participants are asked to perform tasks such as games, ordering objects, searches, etc. in which they subconsciously apply instances of algorithmic methods (*instrumental* knowledge). They are later induced to reflect about the method employed to solve the problem and the reasons for their success (or failure), and to write down their descriptions in natural language. This constitutes evidence of construction of *conceptual* knowledge of algorithms and data structures, according to the general law of cognition. However, in the case that the object on which knowledge is to be constructed is a *program*, some challenges appear, which are inherent to the relevance of the *machine* that executes it, not the *person* who solves the problem. We found the necessity of developing an extension of Piaget's general law of cognition as we identified the need to describe cases where the subject must instruct actions to a computer (da Rosa & Aguirre, 2018). The thought processes and methods involved in such cases differ from those in which the subject performs the actions her/himself. The main objective of the work presented here is to deepen the study of these processes.

2.2. The extended law of cognition

Seymour Papert states in (Papert, 1980) page 28, referring to the programming of a turtle automata, *"Programming the turtle starts by making one reflect on how one does oneself what one would like the turtle to do"*. In other words: "programming an automata that solves a problem, starts by making the student reflect on how she/he does themselves what she/he would like the automata to do".

In order to program an automata to solve a problem, the learners have to establish a causal relationship between the algorithm (she/he acting on objects), and the execution of the program (the computer acting on states). Not only do they have to be able to write the algorithm (the *text*), but also they have to be able to understand the conditions that make the *computer* run the program. The generalisation of Papert's words above can be described as: programming an automata starts by making one reflect on

$$\underbrace{how\ one\ does\ oneself}_{what\ one\ would\ like\ the\ automata\ to\ do}$$

The causal relationship between both rows is the key of the knowledge of *a machine executing a program*. By way of analogy with Piaget's law we describe this relationship in the following diagram

$$\underbrace{C \leftarrow P \rightarrow C'}_{newC \longleftarrow newP \longrightarrow newC'}$$

The conceptual knowledge about the algorithm constitutes what is called *newP* (new periphery) in the diagram. The construction of knowledge about the program is explained by the transitions to new centers (*newC* and *newC'*) which represent awareness of what happens inside the computer. Respectively, the conceptualisation of how the computer executes the instructions of the program (*newC)* and the changes that they impose on the data structures used in it (*newC')*. The diagram describes an extension of the law of cognition to encompass not only algorithmic thinking (first row) but also computational thinking (second row). The relationship between them is indicated with the brace between both lines.

Piaget identified that the construction of knowledge of methods (algorithms) and objects (data structures) occurs in the interaction between C, P and C'. Likewise, we claim that the construction of knowledge of the execution of a program takes place in the internal mechanisms of the thinking process; marked by the arrows between *newC*, *newP* and *newC'*. In other words, the general law of cognition remains applicable to the thinking process represented by the arrows, in both lines of the diagram.

3. Designing the empirical study

The study was conducted with thirteen students from two groups of an initial programming course. Those in the first group (seven students) worked with *C++* and those in the second (six students) worked with *Pascal*. The topics covered so far in the course were the same in both groups, except only for the programming language. The study is divided into two parts and all the activities were carried out by students working individually. The first part analyses the process of constructing *conceptual* knowledge (*inter* stage) from *instrumental* knowledge (*intra* stage) and is based on Piaget's general law of cognition.

The second part investigates the *formal* knowledge construction process (*trans* stage) from conceptual knowledge, and is based on the extension to the aforementioned law. The design of each part is described below relating the activities to the supporting theoretical principles.

3.1. Design of the first part (*intra* → *inter*)

For this part, the student is asked to carry out an activity designed to solve a concrete instance of the linear search problem and, after achieving it, to answer questions aimed at obtaining a precise description of how she/he solves it and why the method is successful. The student is presented with a row of numbered cards on a table and is told that they represent door numbers of houses on a street. The task is to search for a specific person within the row of houses.

Figure 1 – Simulation with houses

Under each card there is a second card with another number, which represents the id number of the person inside the corresponding house. It is assumed that only one person lives in each house and that every id number is unique. The door numbers are ordered and visible to the student, while the id numbers are not ordered and hidden from the student's view (pretending to be the people inside their houses).

Figure 2 – Simulation with cards

The row of cards simulating door numbers resembles an array of integers in a programming language. The door numbers represent the (ordered) indices of the array, while the id numbers represent the (unordered) values stored in their cells. The array is a *computational* representation of the row of houses.

The student is asked to look for a certain id number in the row of cards, in a similar way to being physically positioned at the beginning of the street, in front of the first door. She/he is prompted to go through this process twice; one to search for a number that is in the row and another to search for a number that is not. Because of its nature, each student is expected to successfully solve the task by her/himself, both when the searched number is found inside a door (ending the search at that point) and when it is not (ending after having visited all of the doors).

Once the problem has been solved, each student is asked questions to help them accurately describe the actions performed and explain why her/his method is successful in both cases. It may be necessary to repeat the task so that the student becomes aware of what she/he says in relation to what she/he did, in some cases it could be necessary to pose new questions according to the student's answers. When the description finally matches her/his actions, she/he is asked to put it in writing. This constitutes a first expression in natural language of the linear search algorithm and will be used as a starting point for the second part of the study. In line with the general law of cognition, this process induces the student to become aware of the coordination of the actions carried out (transition from P towards center C) and the changes that said actions impose on the manipulated objects (transition towards center C').

3.2. Design of the second part (*inter* → *trans*)

The passage to the *trans* stage encompasses two aspects. The first is that, being *formal* knowledge, it involves the use of a *formal* language (a programming language). The second is that the execution of the resulting program is not carried out by the student, but by an *external agent*: the computer. Just as the general law of cognition explains the construction process when it is the subject who performs the actions and builds conceptual knowledge about the algorithm, its extension explains it when the actions are instructions executed by the computer. For this part, three activities were designed to guide the student in the passage from the description in natural language to the writing, compilation and execution of a program that solves the problem on a computer.

In the first activity, each student is asked to write a version of the algorithm using *pseudocode*, based on her/his description in natural language. Pseudocode is an intermediate formalism used to facilitate the beginning of the passage to the *trans* stage. It has less rigorous syntax and semantic rules than a formal language, being easier to understand and allowing a first expression of the algorithm, in order to instruct an external agent (different from the student her/himself) to execute it. In this case, the external agent is an imaginary robot (played by the interviewer), which allows the student to visualise the behaviour of the algorithm, leaving for later aspects related to machine execution.

Each student is asked to write several progressive versions of the algorithm, as a way of gradually consolidate the concepts and correcting errors between each version and the next, until finally arriving at a correct one. According to the extension to the general law of cognition, the purpose is to make them aware that actions are now executed not by the student her/himself, but by the external agent (start of transition from *newP* towards *newC*), which imposes changes on objects to successfully reach the solution (start of transition from *newP* towards *newC'*) (for detailed application of the extended law, see for instance (da Rosa & Aguirre, 2018)).

Between each version and the next, *automation* is used. This mechanism involves the interviewer acting as an external agent, working as an imaginary robot that executes the steps while the student observes its behaviour. This is done to induce the student to reflect on errors that may be detected and correct them, posing questions to the student similar to those of the first part. This process is repeated until the robot executes the pseudocode and successfully solves the problem in any case.

In the second activity, each student is asked to write some code snippets to become familiar with the syntax and semantic rules for working with *arrays* in the programming language used by their group (*C++* or *Pascal*) and review other elements of the programming language worked on in class before the study, which she/he will need later to write the linear search program. In addition, this activity helps the student to detach her/himself from the *concrete* instance of the problem (searching for a document in a row of 7 doors) and brings them closer to the more *general* problem of searching for a value in an array of N cells, which they will have to program in the next activity.

As in the first activity, each student is asked to write several versions for each snippet, resorting again to automation until she/he manages to do it correctly and prompting reflection by means of questions in case of errors. The interaction is now with formal object representations of the programming language (an array drawn on paper) rather than with physical objects (numbered cards in a row) that represent objects from the original problem (houses on a street). At this point, automation is used again and the interviewer executes the student's instructions, pretending to be the computer, working on the array.

Automation contributes not only to error detection, but also to the construction of a *general* solution. Although in (Gréco et al., 1963) the jump from specific instances to the general case is investigated by B.Matalon at any stage of knowledge construction, in this study it is especially analysed in the passage to the *trans* stage. Matalon states that in order to construct the concept of *generic* element, it is necessary to carry out a specific action that, by successive repetition, allows the construction of said concept. In this study, the results of Matalon are interpreted by urging the student to work with arrays of different sizes in the snippets, so she/he can build the notion of generic number of elements, using the constant N in the text of the program for the size of the array instead of a specific value (such as 7).

Finally, in the third activity each student is asked to write, compile and run a program that searches for a value in an array of integers. She/he is asked to write it based on the pseudocode version of the first activity, using the syntax and semantic rules reviewed in the second activity. All variables are assumed already declared and initialised. The student must translate the pseudocode steps into formal language instructions. This translation implies a reflexive process, in which she/he must establish a correspondence between the steps carried out on the row of cards and the instructions executed on the data structure that represents it (the array). Like the two previous activities, this one also implies the construction of knowledge on two aspects: use of a formalism (now a programming language instead of pseudocode) and execution by an external agent (now the computer instead of an imaginary robot). The construction encompasses not only the *textual* part of the program but also its *executable* part, and the relationship between the two (see Section 1).

The student writes a first version of the program on paper, and automation is used again on an array drawn on paper to detect and correct errors. Questions are asked aimed at situating their thinking on what the *computer* must do to solve the problem instead of the student her/himself, focusing on issues typical of machine execution, such as, for example, an *out-of-range* index error or a iteration that does not end (*infinite loop*). The whole process is repeated until the student builds a final version that solves the problem correctly and explains how and why. This enables the student's thinking to consolidate the transitions from *newP* towards *newC* and *newC'* (see Section 2).

The activity ends by asking the student to transcribe the text of the program to the computer, compile it and run it at least twice: once to search for a value that is in the array and another to search for a value that is not. Machine execution constitutes the validation of the knowledge built on the *executable* part, since it is the interaction between the subject and the true external agent (the *computer*). The instructions for initialising the array and requesting the value to search are already provided in a source file. The student must complete the missing portion with the linear search code and display a message indicating the result. If a compilation error occurs, she/he is asked to read it and reflect on how to correct it. If an execution error occurs, she/he is asked to go back to the array drawn on paper and automation is repeated, in order to detect the problem, fix the code and recompile and execute the program again.

4. Implementing the empirical study

In this section we include the implementation of the designed activities and a brief analysis of students' work. All students were successful in writing, compiling and running a program that solves the linear search problem in the programming language used by their group (*C++* or *Pascal*).

4.1. Implementation of the first part (*intra* $\rightarrow$ *inter*)

For solving this problem, three fundamental actions must be carried out: the comparison of the searched id number with that of the person within each door, the advance to the next door and their repetition at each visited door. Repetition generates a progressive reduction in the number of doors, which eventually leads to the student seeing one of two possible changes due to her/his actions: a change in the relation resulting from the comparison of id numbers (from *different* to *equal*) or a change in the number of doors that remain to be visited (from being *greater than zero* to being *equal to zero*). In terms of the algorithm, these changes represent its two possible conditions for stopping: when finding a number equal to the searched one or when no more doors are left to visit. As an example, the description in natural language given by one of the students is shown below.

Assuming that you want to find the person with id x, you go through the doors in order, opening them and asking the person behind them for their id number. If it's the one we were looking for, the search ends, otherwise, we go to the next door. If all of the doors are visited and the person with id number x is not found, it is deduced that he/she does not reside behind any of the doors.

The student expresses all three actions, comparison (*asking the person behind them for their id number. If it's the one we were looking for*), advance (*we go to the next door*) and repetition (he describes actions referring to the *doors*, in plural). Regarding the changes, he stops both when he finds the id number (*If it's the one we were looking for, the search ends*) and when there are no more doors to visit (*it is deduced that he/she does not reside behind any of the doors*).

All students wrote similar descriptions, describing both the actions (which constitutes evidence of the transition of thought from P to C) and the changes imposed on objects (evidence of transition from P to C'). As expected at this stage, each description was tied to the *concrete* instance of the problem (searching for a person within a row of houses). From this instance, the more *general* problem of searching for a given value in an array of N values was worked on in the second part.

4.2. Implementation of the second part (*inter* → *trans*)

As explained in section 3.2, the second part comprises three activities. In the first one, each student wrote several versions of the algorithm using a pseudocode, based on her/his own description in natural language. An example is shown below.

```
while (not find id number)
    ask for id number in door
    if it's the one I look for
        stop
    else
        end search
    end
end
```

In this version, the student expresses two of the three actions: comparison (*if it's the one I look for*) and repetition (*while*), but he does not express advancing to the next door (he writes *end search* after *else*). As for the *while* condition, he intends the external agent to stop when finding the searched number: *while (not find id number)*, but not when there are no more doors left. In terms of the extension to the general law of cognition (see section 2.2), he is focused on the desired result of making the external agent find the number (*newP*). He has conceptualised the algorithm after applying it by himself but he must conceptualise the action to be executed when the number is not found in the current door (his thought needs to advance towards *newC*) and the second condition to stop the iteration (needs to advance towards *newC'*). After automation, he notices the first problem and writes a second version, by replacing *end search* with *go to the next door*, but the second problem still remains.

```
while (not find id number)
    ask for id number in door
    if it's the one I look for
        stop
    else
        go to the next door
    end
end
```

The missing condition of the *while* instruction is conceptualised after restoring to automation once again, when trying to find a number that does not exist within the row. He intends to continue searching after the last door has been visited, which prompts him to write a third version.

```
while (not find id number) or (there are no more doors)
    ask for id number in door
    if it's the one I look for
        stop
    else
        go to the next door
    end
end
```

Although his thought has advanced towards *newC'* by including both conditions[1] he needs to reflect

[1]We recall that newC' represents the modifications on data structures, on the one hand, the cards numbers become equals,

about the relation between the semantics of boolean operators and the *while* structure, in order to properly express how to stop. Once again, automation is helpful. When searching again for a number that does not exist within the row, he notices that the condition remains true after having visited the last door, due to the usage of the *or* operator, so he replaces it with *and* and removes the negation in the second condition, resulting finally in a correct solution, as shown below.

```
while (not find id number) and (there are more doors)
    ask for id number in door
    if it's the one I look for
        stop
    else
        go to the next door
    end
end
```

In the second activity, each student exercised the syntax and semantics rules of the programming language used by their group (*C++* or *Pascal*) to work with *arrays* and their integration with other elements of the language necessary for the third activity. Students wrote short code snippets (not included here for space reasons) in which they not only exercised these rules, but also worked with arrays of different sizes defined by a constant N, in order to conceptualise the generic array of N elements from concrete instances, according to Matalon's research (see section 3.2).

In the third activity, each student wrote, compiled and executed a program that searches for a value within an array of N integers, based on the pseudocode version of the first activity. The focus is now on the correspondence between the steps of the pseudocode and the programming language instructions (knowledge about the *textual* part of the program) and on aspects specific to machine execution, not present in the pseudocode, for example those related to the errors mentioned below (knowledge about the *executable* part). As an example, a sequence of versions of the student's work in C++ is shown below, starting from the pseudocode above (the array indices belong to the range 0..N-1).

```
boolean found = FALSE;
int i = 0;
while ((found = FALSE) && (i < N-1)) {
    if (arre[i] == number) {
        found = TRUE;
    } else {
        i = i+1;
    }
}
```

The student establishes a suitable correspondence between the steps in pseudocode and the instructions in the program. He maintains the *while* structure, the conditions are in the same order, and he combines them with *and* (&& in C++), just like in his pseudocode version above. He unifies in one single statement *if* (*arre[i]* == *number*) two separate steps of his pseudocode (*ask for id number in door* and *if it's the one I look for*). His program remains faithful to the behaviour expressed in his pseudocode, which evidences student's construction of knowledge on the *textual* part of the program.

As for the *executable* part, two errors are present. The first is that he uses the *assignment* operator (=) instead of the *comparison* operator (==) in the first condition. The second is a border error in the second condition (he uses < instead of <=, leaving the last array cell unchecked). These type of problems arise specifically at this stage of the process because they involve knowledge construction about *execution on a computer*. Automation is used again, working on an array drawn on paper, allowing the student to detect and correct both errors, resulting in a second (and correct) version shown below, and being finally

and on the other hand, the row of cards becomes empty.

able to explain how and why it works correctly (not included here for space reasons). This constitutes evidence that his thought has continued to advance towards *newC* and *newC'*.

```
boolean found = FALSE;
int i = 0;
while ((found == FALSE) && (i <= N-1)) {
    if (arre[i] == number) {
        found = TRUE;
    } else {
        i = i+1;
    }
}
```

Finally, the student transcribed his code to the computer, compiled it, and successfully executed it. Just like this student, all the others managed to do it successfully as well. For all of them, the passage from pseudocode into a program required fewer attempts (up to three versions) than the passage from the natural language description into pseudocode (up to eight versions). This shows that the start of the process of transforming *conceptual* knowledge into *formal* knowledge was more difficult for all students. According to the extension to the general law of cognition, the beginning of transitions from *newP* to *newC* and *newC'* is a key milestone within the entire process, since at this point each student's thought needs to restructure in order to shift from knowing how to apply the algorithm her/himself towards instructing an external agent to perform the task. The difference in the programming language (*C++* or *Pascal*) made no difference in this regard at all. Below is the program written by a student from the second group, using *Pascal* instead of *C++* (in this program, the array indices belong to the range 1..N).

```
noidnumber := False;
count := 1;
while (noidnumber = False) and not (count = N+1) do
begin
    if (arre[count] <> number) then
        count := count + 1
    else
        noidnumber := True;
end
```

5. Conclusions and further work

This paper presents an empirical study that uses the linear search problem to investigate the process of construction of knowledge about programs, based on a theoretical framework we have constructed from Piaget's principles of a triad of *intra*, *inter* and *trans* stages (see Section 2). It constitutes a detailed case study that shows how our theoretical framework can be used to explain the complete process (from *instrumental* to *formal*) of knowledge construction on programming concepts.

The study analyses the role of students' construction of knowledge previous to any formalisation in the passage from the *intra* stage into the *inter* stage, regulated by Piaget's general law of cognition (see Section 3.1). The passage from the *inter* stage into the *trans* stage, is regulated by our extension of Piaget's law. The use of an *intermediate formalism* (pseudocode) and the mechanism of *automation* are introduced to help students construct knowledge about the program both as a *textual* and an *executable* object (see Section 3.2).

Several lines of future research arise, some regarding construction of knowledge on other algorithmic problems (such as *counting*, *filtering*, *ordering*, etc.) and data structures (such as *linked lists*, *binary trees*, etc.). Other lines of future work involve analysing knowledge construction when other programming paradigms are used for the formalisation at the *trans* stage, for instance, functional programming, where recursive algorithms play a central role. It is expected that the results from such lines of future research will enable the development of pedagogical guidelines to introduce programming concepts.

6. References

da Rosa,S. & Chmiel,A. & Gómez, F. (2016). Philosophy of Computer Science and its Effect on Education - Towards the Construction of an Interdisciplinary Group. *Special edition of the CLEI Electronic Journal (see http://www.clei.cl/cleiej/), Volume 19 : Number 1 : Paper 5.*

da Rosa, S. (2010). The Construction of the Concept of Binary Search Algorithm. *Proceedings of the 22th Annual Psychology of Programming Interest Group Workshop, Madrid, Spain*, 100–111.

da Rosa, S. (2015). The construction of knowledge of basic algorithms and data structures by novice learners. *Proceedings of the 26th Annual Psychology of Programming Interest Group Workshop, Bournemouth, UK.*

da Rosa, S. (2016). Preconceptions of novice learners about program execution. *Proceedings of the 27th Annual Psychology of Programming Interest Group Workshop, Cambridge, UK.*

da Rosa, S. (2018). Piaget and Computational Thinking. *CSERC '18: Proceedings of the 7th Computer Science Education Research Conference*, 44–50. `https://doi.org/10.1145/3289406.3289412`.

da Rosa, S., & Aguirre, A. (2018). Students teach a computer how to play a game. *LNCS of The 11th International Conference on Informatics in Schools ISSEP 2018* .

da Rosa, S., & Chmiel, A. (2012). A Study about Students' Knowledge of Inductive Structures. *Proceedings of the 24th Annual Psychology of Programming Interest Group Workshop, London, UK.*

Gréco, P., Matalon, B., Inhelder, B., & Piaget, J. (1963). *La Formation des Raisonnements Recurrentiels.* Les Etudes Philosophiques, 18(4). Presses Universitaires de France.

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

Piaget, J. (1964). *La prise de conscience.* Presses Universitaires de France.

Piaget, J. (1974). *Success and Understanding.* Harvard University Press.

Piaget, J. (1975). *L'équilibration des Structures Cognitives, Problème Central du Développement.* Presses Universitaires de France.

Piaget, J. (1977). Genetic Epistemology, a series of lectures delivered by Piaget at Columbia University, translated by Eleanor Duckworth. *Columbia University Press.*

Piaget, J. (1978). *Recherches sur la Généralisation.* Presses Universitaires de France.

Piaget, J., & Garcia, R. (1980). *Psychogenesis and the History of Sciences.* Columbia University Press, New York.