CLEI ELECTRONIC JOURNAL, VOLUME 23, NUMBER 1, PAPER 5, APRIL 2020

A research model in didactics of programming

Sylvia da Rosa
UDELAR, Facultad de Ingeniería, Instituto de Computación
Montevideo, Uruguay, 11300
darosa@fing.edu.uy

and

Federico Gómez
UDELAR, Facultad de Ingeniería, Instituto de Computación
Montevideo, Uruguay, 11300
fgfrois@fing.edu.uy

Abstract

This paper presents a research model in didactics of programming elaborated within the theoretical framework of the epistemological theory of Jean Piaget. That theory explains the construction of scientific knowledge based on empirical studies made by Piaget over many years. The model arises from the analysis of the results of the application of principles of the theory, especially the triad of *intra-inter-trans* stages, to the empirical study of the construction of the concepts of algorithm, data structure and program. The elaboration of the model is a contribution to the development of the didactics of programming and, in general, of the didactics of computer science, since the model can be used in other computer science topics. Didactics is a specific area within computer science, with its own foundations and methods, which studies in depth topics related to education in the discipline. Two empirical studies about the construction of knowledge of algorithms and data structures, and of the corresponding programs as executable objects, are briefly described to illustrate the model. Both examples use a search algorithm (binary and linear) and the implementations are in the programming language C.

Keywords: didactics of programming, constructing knowledge, algorithms and data structures, programs.

1 Introduction

What is and what is not Computer Science? In [1] Holmboe, McIver and George give some guidelines about those questions and put on the table interesting issues. They give a possible and open-minded definition of Computer Science (hereinafter CS) as *the collection of scientific disciplines oriented towards the electronic or digital storing and processing of information*, and emphasise that CS constitutes a scientific or academic discipline as opposed to a craft. The authors mean that some areas related to using computers or technology, are not CS. Some of these are, for instance, technological literacy (use of computer applications, such as word processing or spreadsheets) or technology for education (use of technological tools, such as audiovisual resources for education or online platforms for interaction between teachers and students). Historically, these areas have been present in pre-university education, often being confused with computer science, while computer science education has been mostly restricted to the tertiary or higher context, more specifically to careers which purpose is the training of technicians or professionals linked to computers, such as technical, undergraduate or engineering careers.

In the same way, Dowek states in [24] that computer science - in the French and German tradition the term informatics is used, while in the English tradition the expression computer science is used - is a scientific discipline that studies algorithms, data, languages and machines, and is different from other areas linked to the use of computers and technology. In recent years, academics and researchers from various countries have -with greater or lesser success- have promoted the idea that education in computer science is not only important for the training of

professionals in the discipline, but that it is also fundamental as part of the general training of any individual. This is promoted by various authors from organizations such as Computing at School (CAS) in the United Kingdom [2], CODE in the United States [3], and the Computer Science Teachers Association (CSTA) [4] or Association Enseignement Public & Informatique [5], which brings together teachers from different countries. Already in the 80s, projects for children and high schools were developed. Especially Papert poses in [6] that: *children can learn to program and that learning to program can affect the way they learn everything else*. In [7], at the beginning of the 2000s, Dowek expressed about secondary education: *L'apprentissage de la programmation et de l'algorithmique est de nature à apporter beaucoup aux lycéennes et lycéens dans leur développement intellectuel, car il permet un travail par projets et demande de mettre en application des connaissances acquises. Et également car il permet de construire un pont entre le langage et l'action et montre l'utilité de la rigueur scientifique (learning of programming and algorithms has much to contribute to high school students in their intellectual development, among other things because it enables them to work in projects and requires to apply acquired knowledge. Likewise, it enables to build a bridge between language and action and show the utility of scientific rigor* - translation of the authors).

Those questions open new problems, where three of the most relevant are: a) what does it mean to learn computer science in general, and programming in particular? b) What kind of academics should answer the previous question? c) What teachers are qualified to teach CS to pre-university students? Searching answers to those questions leads to a considerable development and strengthening of computer science education and of doing research on *how* to educate in programming in childhood and adolescence. At the same time, there is a general consensus about that the origin of most of the difficulties in learning programming lies in the background of the students. This is supported by teaching experience in introductory courses in CS. In this sense, investigating novice students' knowledge of programming has become an issue with high impact in higher education in computing. Researchers of computer science of several countries' Universities have made that issue their own, as can be seen in the variety of conference proceedings in the area of recent years (SIGCSE, ITiCSE, ACE, KOLI, WiPCSE, ICER, CSERC, ISSEP, PPIG, among others), and journals (e.g. Computer Science Education, ACM Transactions on Computing Education, and Informatics in Education).

1.1 Doing research in didactics of programming

It is worth mentioning that the term didactics is not commonly used in English education and the term pedagogy encompasses almost all educational issues. Paul Andrews points out in [25] that *in the European education the term didactics comprise the strategies to subject teaching and learning, which may vary from one subject to another. Didactics also acknowledges theories of teaching and learning but from the subject-specific perspective.* Analogously the term *Computer Science Education* (hereinafter CSE) is used in English for the European *Didactics of Informatics.* A definition is given by Holmboe, McIver and George in [1] as *the subject specific educational research for the subject computer science*. That means that, as in any other area of knowledge, informatics has an area where computer science professional academics investigate didactic problems.

In [1] the authors enumerate several studies in the learning of Computer Science like: *New, untested ideas*, *Reports from the trenches* (specific experiences in the classroom) or *Empirical Studies* (experimental works aimed at analyzing difficulties and behaviours of students when learning specific programming topics). In [8] Hubwieser, Armori, Giannakos and Mittermeir provide a comparison of the status of issues related to CSE in primary and secondary schools, analysing some papers of authors from different regions (UK, New Zealand, USA/Israel, France, Sweden, Georgia/USA, Russia and Italy). They conclude that in the majority of those papers *proper teacher education seems to be one of the most critical factors for the success of rigorous computer science education, and that programming in one form or another, seems to be absolutely necessary for a future oriented CSE.* They add that *despite the fact that there have been several and important developments in K-12 CS education during the last years, there is still much that can be done, especially in K-12 educational systems with low CS subject integration.*

It seems quite obvious that the lack of knowledge on learning and cognition is one of the main problems for doing research in didactics of informatics. Holmboe, McIver and George state in [1]: *there has been a lack of reference to pedagogical theory, underlying most past research studies. This has resulted in a failure to provide teachers with "pedagogical content knowledge", critical to gaining useful insights into cognitive and educational issues surrounding learning*. In the same way, Peyton Jones et al state in [9]: *one reason for the lack of expertise in computer science teaching is the background of the teachers.*

The question is then to adopt an approach encompassing both the specific knowledge of the discipline and its didactic issues. The paradigmatic example is the didactics of mathematics, which has been developed as a specific

discipline in mathematics, with its own investigations carried out by mathematicians specialized in didactics. Note that this is different from the joint work between academics of a discipline and professionals in the field of education.

1.2 Pedagogical Content Knowledge

Among CSE researchers there is consensus on adopting Shulman's definition [10] of pedagogical content knowledge (hereinafter PCK) as: *the ways of representing and formulating the subject that make it comprehensible to others*. In their CSE research, Saeli, Perrenet, Jochems and Zwaneveld [11] specify the PCK as: *concept that combines the knowledge of the content (e.g., maths, informatics, etc.) to the knowledge of the pedagogy (e.g., how to teach maths, how to teach informatics, etc.), giving insights into educational matters relative to the learning and teaching of a topic. Teachers with good PCK are teachers who can transform their knowledge of the subject into something accessible for the learners*. (Note that for the same reasons explained by Paul Andrews in [25], the *Pedagogical Content Knowledge* should be called *Didactic Content Knowledge)*.

In [1] Holmboe, McIver and George synthesize and apply the ideas for the CSE case stating that: *The aim (of CSE) should rather be to describe the different ways in which students come to understand, or not to understand, the subject matter. These descriptions, accompanied by knowledge of general pedagogical theory, epistemology and solid subject knowledge will make the foundation for answering the traditional* ***didactical questions of why, what, how and for whom to educate in CS***. They also state that research needs to be supported by a theoretical framework, since it cannot be reduced to observations made from isolated experiences or opinion issues, but they need to be treated with the same rigor as any other research activity carried out in a scientific context.

For the development of an adequate theoretical framework, these authors point out that: *looking to the variety of work being published in more established fields (e.g. science education, mathematics education and teaching and learning of foreign languages) may give several useful pointers to researchers in computer science education* [1]. Hence they extend their definition of CSE as: *The academic discipline computer science education consists in focusing research on the application of principles from educational-related disciplines -pedagogy, epistemology, curriculum studies and psychology- to the teaching and learning of the scientific discipline computer science as a school subject* [1]. They add that the strong connection with educational-related disciplines constitutes the theoretical argumentation of the research as a means of providing evidence of its effectiveness.

It can be concluded that research in CSE comprises two fundamental issues: researching the construction of computer knowledge and developing teaching guidelines so that the results of the research enable teachers to acquire PCK. This paper addresses the first of the two questions in the case of programming knowledge. The authors of this paper have adopted Piaget's epistemological theory [26] as a theoretical framework for research and Guy Brousseau's theory of situations [32] as a model for didactic application.

1.3 Defining a theoretical framework for didactics of programming

Within the framework of the four fundamental questions - ***why, what, how and for whom to educate in CS*** - this article specifically addresses the construction of knowledge about basic algorithms, data structures and programs (*what*) and their introduction into secondary education (*for whom*). There is a consensus among researchers about the benefits of such learning for all students, not only for those who will continue with higher studies in computer science, as stated by Dowek in [7] and Papert in [6] (*why*).

Regarding the question of *how*, most of authors interpret the question as *how can the teachers teach such item in a better way?* Therefore, they propose teaching strategies or methodologies, for instance, using fundamental concepts as Schwill states in [12], promoting abstraction as Wing states in [13], or through projects, as Dowek states in [7]. On the contrary, the research presented in this article, takes the question of *how* from the student's perspective, that is, *how do students learn programming concepts?* This formulation is a particular case of the problem of how do students learn. Piaget's epistemological theory gives satisfactory explanations about the construction of knowledge, starting from the knowledge people have ***before*** any formalisation. By applying Piaget's theory to the construction of knowledge of algorithms, data structures and programs, the authors of this article have defined a theoretical framework for doing research in didactics of programming.

In contrast to several proposals to help students in the learning of programming involving the use of some programming language or computer tool [14-21], the approach behind the motivation of the authors' proposal is based on observations of situations in day-to-day life in which people successfully use methods to solve problems or

perform tasks such as games, ordering of objects, different kinds of searches, mathematics problems, etc. In such situations an action or a sequence of actions is repeated until a special state is reached, which can be solved easily by a straight-forward action. People's descriptions include phrases like *I do the same until* and *now I know how to do it*, referring to cases where they use the same method and arrive at the easy-to-solve special state respectively. These descriptions are related to programming in the sense that repeating actions until a special case is reached is formalized by recursive or iterative program instructions.

These observations lead to formulate questions such as *does there exist any connection between the 'knowing how to' (instrumental knowledge) revealed by people solving problems and formal algorithms? If there is, what is the nature of this connection and what is the role of the instrumental knowledge in the learning process? How is this instrumental knowledge generated and how can it be transformed into conceptual knowledge? How can the algorithms that the students learn to use be taken into account in the learning of programming? Will the answer to these questions help in improving the teaching and learning of programming and how should this be done?* The approach of the research arises from the above observations and questions, and from studying the theory of Jean Piaget that explains the construction of knowledge and the evolution of cognitive instruments from the interaction of the subject (his/her algorithms) with the objects (data structures).

The article is organized as follows: in section 2, the main principles of Piaget's theory are described; in section 3 the model is introduced and in the subsections 3.1 and 3.2 the main points of the passage from the *intra* to the *inter* stage are described through an example. In subsection 3.3, the construction of general solutions is summarized. In section 4, the main point of the passage from the *inter* to the *trans* stage, namely the construction of formal knowledge, is developed in detail, including extracts of an empirical study in subsection 4.2. Subsection 4.1 describes the extension of Piaget's general law of cognition, a theoretical contribution of the authors to encompass the construction of knowledge about programs. Section 5 contains some conclusions and further work and finally the references are included.

2 Main theoretical principles

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 [26].

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) [27-30], 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 [31] the authors present a synthesis of Piaget's epistemological theory and a new perspective on his explanations about constructing knowledge. They investigate the possible analogy between the mechanisms of psycho-genetic development concerning the evolution of intelligence in children, and socio-genetic development concerning the evolution of the leading ideas and theories in some domains of science. Throughout the chapters the authors present striking examples of this analogy in relation to the history of geometry, algebra, mechanical and physical knowledge in general.

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 generalized elements and their transformations (*trans* stage), integrating the constructions of the previous stages as particular cases.

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 [26]. 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 acquired before their formalization. This problem was deeply studied by Piaget in his experiments about genetic psychology. From these he formulated a general law of cognition [28,29] governing the relationship between know-how and conceptualization, generated in the interaction between the subject and the objects that he/she 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 his/her actions (C in the diagram), the modifications 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 the general law of cognition constitutes a first step towards the construction of concepts.

Piaget also describes the cognitive instruments enabling these processes, which he calls reflective abstraction and constructive generalization [28,30]. Reflective abstraction is described as a two-fold process: in the first place, it is a projection (transposition) to the plane of thought of the relations established in the plane of actions. Second, it is a reconstruction of these relations in the plane of thought adding a new element: the understanding of conditions and motivations. The motor of this process is called by Piaget the search of reasons for success (or failure).

Once a particular method is understood, subjects' reasoning attempts to generalize what has been successfully constructed to all of the situations, by means of inductive generalization. Deductions or predictions are extracted from observations of the new objects. A process of inferences and reflections about the subject's actions by means of constructive generalization gives raise to new methods and new knowledge.

Throughout several years of study about the learning of different basic algorithms and data structures, the authors have been developing a research model in didactics of programming, which is described in the next section.

3 A research model in didactics of programming

Other approaches have been developed within a Piagetian conceptual framework derived from genetic psychology, where one of the most notable is the new-Piagetian theory [22,23]. The approach is supported by Piaget's early empirical work focused on children, which is the most known part of Piaget's work. On the contrary, the model presented here is based on one of the most important concepts of Piaget's theory, that is, the triad of stages *intra-inter-trans*, introduced by the authors in [31] as a synthesis of their *epistemological* work.

In previous papers [33-37] the authors of this paper described the investigations conducted to know about the passage:

- from an *intra stage*, in which the knowledge is instrumental, that is, in the plane of actions (the students pursue a result but are unaware of how they achieve it)
- to an i*nter stage* giving rise to conceptual knowledge, that is in the plane of thought (the students give accurate descriptions of how they did it and why they succeeded, being aware of the coordination of their actions and the transformation of objects).

One of the goals of this paper is to describe the research about the passage from earlier stages above to:

- a *trans stage* of formal knowledge (where the students are able to express general algorithms in given formalisms and modify their knowledge to solve similar problems).

Instantiating the triad of stages of Piaget and Garcia to the construction of knowledge about programming means constructing a research model in didactics of programming.

3.1 Applying the model

In order to illustrate the application of the model, two examples of investigations conducted with students are included. In the first one (section 3.2), they were entering university students or enrolled in the final year pre-university. Therefore, they had no experience with programming (in Uruguay Computer Science is not part of High

School curriculum). The students of the second example (section 4.2) were studying the first year of the computer engineering degree at university and had little experience with programming.

Regarding the methodology of research, the passage from the *intra* to the *inter* stage is investigated by means of conducting individual interviews, in the sense of Piaget's studies of genetic psychology, used in previous empirical studies. The students are asked to solve problems or tasks which are instances of classical and basic programming problems such as ordering, searching or counting elements. The problems are well known and *automatically* solved by the students, that is, the students are unaware of how they did it and why it works. In other words, their thought is at the peripheria (P), see section 2. In the interviews they are encouraged to think about both the coordination of their actions (towards C) and the modifications that these impose to the object (towards C'). The interaction making the passage from P to both C and C' is possible by means of the cognitive instrument of reflective abstraction (section 2): the successfully done actions are transformed into operations, adding a new element: the understanding of conditions and motivations.

All the research episodes were recorded and/or filmed and students wrote down their responses.

3.2 The study of the passage from *intra* to *inter*

The study[1] consists in conducting interviews to ten entering students of an introductory course of programming. The students are required to look up a word in a dictionary and to explain, in natural language, how they did it and why they succeeded. The problem is an instance of the general problem of searching an element in an ordered list and the solution is an instance of the algorithm of binary search. This task is adequate for the investigation because of the following characteristics:

- the algorithm of binary search is commonly applied in solving this task, that is to say, all students know very well the application of this algorithm to this particular case, and all of them succeed in searching a word,
- the task is (almost) automatically done, what gives the opportunity of analyzing the grasp of consciousness and conceptualization in detail from the origin of the process,
- no numeric domain is involved and the role of school is minimal which diminishes the influence of preconceived ideas,
- this algorithm is one of the most important methods of searching and it is taught in all courses of programming.

It is expected that the students solve the problem using the algorithm of binary search, without being aware of what that means, that is, their thought corresponds to the state called the peripheria, P, in the general law of cognition. In other words, the students focus their attention in the result of the task, what is typical of the *intra* stage. To help students to construct knowledge about the relationships between objects and actions, characteristic of the *inter* stage, they are asked about:

- the structure of the dictionary as an ordered list of words, (students' thought moves towards the centre C', questions 1 to 4 bellow),
- the actions that compose the algorithm: choosing a word, comparing words and a new instance of the search itself, (students' thought moves towards the centre C, questions 5 and 6 below),
- the reasons for success: each new instance of the search is applied on "smaller" parts of the dictionary (do the same), all the smaller parts hold the property of containing the searched word (invariant), the search ends when a special case is achieved (termination), (students' thought moves towards both centres C and C', questions 7 and 8 below).

Questions 1 to 4:
Q1: What is a dictionary?
Q2: Knowing that a word, for example 'cat', is in the dictionary and also in this novel, where do you think that it will be simpler (easier and more quickly) to find it?
Q3: Why?
Q4: What makes the difference then between a dictionary and any other book?

1 The study is partially described here, the complete version can be found in [35].

The first question is answered by almost all students saying something about the role of the dictionary (for instance, “it is a book to search words”). The rest of questions are aimed to induce the student to reflect about the structure of the dictionary as a list of words alphabetically ordered. It is a requisite for the *inter* stage because it is the existence of this order relationship that determines the method of searching.

Questions 5 and 6:
Q5: Look up the word cat in the dictionary.
Q6: Describe, step by step, how you achieved it. (Eventually, ask them to do it again).

The answers to question Q6 reveal that the students have a very weak conceptualization about their method of searching a word. The students apply binary search as expected, but they are not aware of the different actions they do to achieve the result. For example, they automatically say “*I searched the word*”, which is a response to the question “*what did you do*” and not to “*how did you do it*”. The question aims to interrupt the automatic actions and to mentally identify and coordinate other actions: to choose a word, to compare it with the searched word, to make a decision according to the result of the comparison, to do the same. This coordination is characteristic of the inter stage.

Questions 7 and 8:
Q7: Knowing that a word is in the dictionary, is it always possible to find it?
Q8: Why?

The answers to these questions can be classified as: a) the reasons lie with the objects ("*... because of the order in the dictionary*" or “*... because of the alphabetic order of the letters*"); b) the reasons lie with the actions ("*... because of my systematic actions*" or "*... because I do always the same*"). To make students realize that the reasons for success lie both in their actions and in their modifications of objects, they themselves have to experience the need for the existence of a base case (or more). To effectively help them with this difficulty they are asked to use a method that never ends, to make the students become aware of the sequence of states of the object, each time getting “smaller” until a special state is reached, as a reason for success.

This kind of questions plays an essential role in the conceptualization of algorithms and data structures (*intra* to *inter* stage), that is to say, in cases where an individual solves a problem. Investigating the passage from the *inter* to the *trans* stage - described in the next section - consists in designing questions to cases where tasks or problems are intended to be solved by a computer (conceptualising of programs).

3.3 Constructing a solution of a general problem

The passage from the *inter* to the *trans* stage basically involves two points: the construction of a solution of a general problem from a given instance, and the construction of formal knowledge, that is, the formulation of the concepts in a given formalism. Both constructions are part of the same process towards the *trans* stage.

Regarding the first point, the generic element problem has been analyzed by Benjamin Matalon in a chapter entitled “Recherches sur le nombre quelconque” in [27]. Matalon addressed the problem of making the leap from particular cases to general ones and introduced variables for their reference. For example, he explained that Fermat made his arithmetic demonstrations using a particular number, but treating it as a generic number, for example, the number 17. If none of the specific properties of the number 17 were involved in the demonstration, then the demonstration could be considered valid for all numbers. Matalon added that in geometry, when a property is to be proven and the statement is "given a generic triangle" a particular triangle is drawn, avoiding right triangles, equilateral triangles or isosceles triangles, and not involving any particular properties of the triangle in the demonstration of the property. Among other things, Matalon concluded that to construct the concept of the "generic" element, it would be necessary to perform a generic action, that is, the repeated action to build a generic element.

A complete description of applying Matalon’s ideas to CSE can be found in [37]. Here a brief explanation is included. Starting from students’ description of how they solved the problem (in natural language), they are asked to give oral instructions to a robot, played by the teacher or another student, who tries to solve the task by following the instructions. The role of this step of automatization is crucial to help students to detach themselves from particular cases: the robot acts until a sentence of the form ... *then I continue the process* ... or *I continue in the same way* … appears, which the robot is not able to follow. Further questions need to be posed to encourage the students

to give more precise and general descriptions (Matalon's generic actions), until they come to a description that the robot is able to perform, that is to say, a general algorithm (Matalon's generic element).

4 The construction of formal programming knowledge

The construction of formal knowledge is the process of constructing representations of the concepts in some formalisms, different from natural language. In the case of programming, it is about to formalize the algorithms in programming languages giving rise to programs. This is the interpretation in the model of what Piaget and Garcia call *thematized knowledge* in [31] and is one of the main issues of the passage to the *trans* stage.

The construction of formal knowledge is regulated by 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. The authors of this paper have extended Piaget's general law of cognition to account for the need to describe cases where the subject must instruct an action to a computer [39,40]. In order to program an automata to solve a problem, the learners have to establish a causal relationship between the algorithm (that he/she apply on objects), and the elements relevant to the execution of the program (the computer acting on memory states). Not only do the students have to be able to write the algorithm (the text) and represent it using a programming language but also they have to be able to understand the conditions that make the computer run the program [38]. The new law is briefly described below.

4.1 The extended general law of cognition

In the cases where the subject must instruct an action to a computer, the thought processes and methods involved in such cases differ from those in which the subject instructs another subject, or performs the action him/herself. An extension of Piaget's general law of cognition has been elaborated [39,40] to take into account the specificities of the subject instructing a computer to solve the problem.

By way of analogy with Piaget's law that relationship is described in the following diagram

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

The causal relationship between the first row and the second row is the key of the knowledge of a machine executing a program. It is indicated with the brace in above diagram, where newP is characterised by a periphery centred on the actions of the subject and the objects he/she acts on. The centres newC and newC' represent awareness of what happens inside the computer: newC of the execution of the program instructions and newC' of the undergone modifications of the representation of data structures The diagram describes the situation in which the subject reflecting on his/her role as problem solver becomes aware of how to do to make the computer solve the problem [39].

According to Piaget, the authors of this article identify that the construction of knowledge of methods (algorithms) and objects (data structures) occurs in the interaction between C, P and C'. Likewise, they 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 pictured above.

4.2 The study of the passage from *inter* to *trans*

The study of the passage from the *inter* to the *trans* stage is described through an example using an algorithm of linear search, which implementation in a programming language is simpler than the one of binary search. The problem is to search an element in a sequence of not ordered elements and the goal is to study the transformation of the informal algorithm in a program in the programming language C. In order to start the study from correct descriptions in natural language of an instance of the algorithm of linear search (first step in the conceptualization), a row of numbered cards is presented to students, simulating doors of houses (that can be represented by an array). On the other side of each card there is a number simulating the identification of a person living in that house. The identification numbers are not ordered. The students are asked to search a given identification number in the row of cards (houses).

For the study, twelve students of the first year of Computer Science degree were selected. At the time of the study they had basic knowledge on the programming language C, especially variables, arrays, simple instructions and control structures, both selection and iteration.

In the same way as in the binary search study (section 3.2), all students were able to successfully perform the task on the row of cards (*intra* stage) and answer the questions proposed in the followed interview (*inter* stage) (not included here). In the following the work of one of the students is used for exemplifying (all works have been similar). His description after performing the task on the row of cards is:

> *"I go to the first door and look for the desired number. If I find the number there, I select that door and do not go on with the rest, if I don't find it, I go to the next door and repeat the same process until I find the number. In case I don't find it after I finish searching through all of the doors, I leave."*

From this kind of description, the goal is to help the students in writing and executing a program for the algorithm of searching a value in an array of N elements. As a first step, a pseudo code is used, in which the semantic of control structures (while, for, if-then-else) are reviewed prior to the writing of the pseudo code. The pseudo code has also the advantage of acting as an intermediate formalism allowing the generation of more instances of the generic action driving to the generic element (the program) according to Matalon (section 3.3). The first version that the student writes is:

```
while the id number is not found
        ask for the person's id number
        if it's the one I'm looking for
                write down the door number where the person was found
        else
                finish
        end
end
```

Note that students usually write a first version focused on the result of the requested task, (periphery P according to the general law of cognition), which in this case is evidenced by the fact that the need to find the identity card, makes them forget to take into account the actions to be performed: he writes *finish* after *else*, despite he mentioned the actions in his description: *if I don't find it, I go to the next door and repeat the same process until I find the number*.

The process consists of executing the written instructions, checking for errors and writing new versions. It is a slow process done in several steps. First, the student notes that the algorithm is not correct because no iteration is indicated. He says: *"it fails because I want to continue"* and he writes the following version adding the action *go to the next door*:

```
while the id number is not found
        ask for the person's id number
        if it's the one I'm looking for
                write down the door number where the person was found
        else
                go to the next door
        end
end
```

Asked about what happens if the identification number is not found he verifies that in that case the search never ends. He writes a new version (note that the original description is correct in the sense that it contains that condition - *In case I don't find it after I finish searching through all of the doors* -):

```
while the id number is not found OR there are more doors to visit
        ask for the person's id number
        if it's the one I'm looking for
                write down the door number where the person was found
        else
                go to the next door
        end
end
```

As can be seen, there is still a failure related to the coordination of the actions of verifying if the card has been found and if there are still doors. The student has joined those actions by means of the OR disjunction, when the correct way is to do it by means of the AND conjunction. This is one of the purely formal aspects, which depends on the semantics of the logical structures and the instruction WHILE, that the student knows. By re-running the algorithm *focusing on the coordination of both actions*, he can realize that he continues searching even after finding the searched ID, and correcting the version obtaining a correct one:

```
while the id number is not found AND there are more doors to visit
        ask for the person's id number
        if it's the one I'm looking for
                write down the door number where the person was found
        else
                go to the next door
        end
end
```

By reflecting about the reasons of success in the last version, the student becomes aware that the search ends because either the number is found or *the sequence of doors is empty.* That is to say, the reasons of success lie in the modifications imposed to the objects (the data structures) by the actions (the instructions), that is, in the dialectic relationship between actions and objects (section 2).

Note that students build their program by matching the concept they have correctly expressed in natural language with the terms imposed by the cultured object that is a formalism, first in its pseudocode version and later in the programming language. By means of manually running each version and verifying the presence or not of errors they perform the generic actions necessary for constructing the generic element (the program) (section 3.3).

However, this is the *textual version of the program*: the errors detected and corrected, arise from the (automated) execution of a human being. In the construction of a program as an *executable object*, a similar interaction occurs, but where the execution is performed by the computer. This fact introduces questions that are typical of computer execution and that were not present in the pseudocode versions. Some of them are:

- Representing data structures as memory objects, following the rules of the programming language syntax, (such as arrays used in this case).
- Trying to access an array with an index outside its declared range, producing a range error.
- Producing stack overflow error, because of not actualizing the control variable of an iteration and therefore entering an infinite loop.
- Incorrect managing of edge cases (for instance, not considering first or last cell of an array because of inadequate control of the array index in an iteration).

The students are asked to write a program for searching a given integer value in a not ordered array of integers, using the last version of the algorithm previously written in pseudocode. The kind of similarities and differences of this problem with the problem of the sequence of doors makes it adequate for these students (the cognitive rationale is described in [37], section 3.2).

The students know the basis of the programming language C, therefore this is the language used in the study. Both the values in the array (**arr**, indexes **0..N-1**) and the searched value (**idNumber**) are given. The program uses a boolean variable (**found**) to determine if the value is in the array or not.

One of the most common mistakes of students in search algorithms in an array is "going out of range", that is, if the range of the index of the array is, say, between 0 and N-1, the index variable reaches the value N. This is an example of errors that occur because it is the computer that executes the program and not a human being (second item of the list above). Therefore, this mistake is not present in the pseudocode, but appears when the student writes the program code, although the error arises at execution time. In terms of the general law of cognition, it can be said that in writing the pseudocode, thought processes are regulates by Piaget's general law (first row of the diagram in section 4.1) but in writing the program the student has to be induced to think in a computational way, that is, he/she has to make the correspondence between the first row (how do I do to solve the problem) and the second row (how do I do for the computer to solve the problem) in the diagram of section 4.1. In other words, the process of thought is regulated by the extended general law of cognition. To help students in making such correspondence, they are asked to automatically run the algorithm on an array drawn on the paper.

In the following, the analysis of the versions of the program written by three students illustrates the point:

Student 1, version 1

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

Note that in the first version student 1 declares a boolean variable but does not use it in the subconditions of the while structure. All students who did not use the boolean variable in the while condition, wrote the subconditions in the order in which student 1 did, that is (arr [i]! = IdNumber) && .... In this way it is very likely that there will be an error of going out of range, if i exceeds the value N – 1. To correct the mistake some students changed the order of the subconditions of the while - see the code of student 3 - others used the boolean variable in the subcondition of the while, as student 1 did in his second version shown below:

Student 1, version 2

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

Student 2, version 1:

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

In this case, the student first takes into account that he should pay attention to the range of the index i of the array, but considers the value N that does not belong to that range in C. He easily obtains the correct version by changing the condition (i <= N) by (i <N):

Student 2, version 2:

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

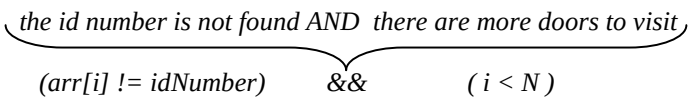
Student 3, version 1

```
int i = 0;
boolean found = FALSE;
while (arr[i] != idNumber && i < N) {
   if (arr[i] == idNumber) {
      found = TRUE;
   } else {
      i++;
   }
}
```

Student 3, version 2

```
int i = 0;
boolean found = FALSE;
while (i < N && arr[i] != idNumber) {
   if (arr[i] == idNumber) {
      found = TRUE;
   } else {
      i++;
   }
}
```

In version 1, student 3 writes the subconditions of the while as they appear in the pseudocode, that is:



The student took into account that she should not consider the case i = N (puts i <N) but forgets that the actions are executed by an automaton and therefore, when i = N, the instruction *(arr[i] != idNumber)* is executed and produces an error of going out of range (arr [N] is invalid).

From the perspective of algorithmic thinking, regulated by Piaget's general law of cognition, that is correct, since the AND operator is commutative. But from the perspective of computational thinking, regulated by the *extended* general law of cognition, it is necessary to consider how the computer "thinks". In that case, the AND operator is not commutative, since in the vast majority of programming languages the short-circuit evaluation is used to implement the semantics of the AND operator. Therefore the order of the operands is relevant to the result.

The reflection on the error that occurs when executing the code places the student's thinking in the second line of the diagram of the extended law of cognition. The comprehension of the reasons of success as the dialectic relationship between the operation of comparing the integer values **arr[i]** and **idNumber**, and the modifications of the array cells in which that operation acts **(i < N)**, allows the student to correct the error of going out of range, as shown in version 2. In terms of the extended general law of cognition that means to go from newP to newC and newC'.

However, by focusing attention on the error to be corrected, it keeps the variable found in FALSE when idNumber is found. Finally, she gets a correct version using the boolean variable in the condition:

Student 3, version 3

```
int i = 0;
boolean found = FALSE;
while (i < N && found == FALSE) {
   if (arr[i] == idNumber) {
      found = TRUE;
   } else {
      i++;
   }
}
```

All students use a boolean variable, without considering that an expression such as arr [i] == idNumber is of the boolean type and therefore a boolean variable in the while condition is not necessary, but must be used to return the result. They could write, for example:

```
int i = 0;
boolean found;
while (i < N && arr[i] != id) {
      i++;
found =  (i < N)
}
```

The abuse of boolean variables may have their origin in preconceived ideas about control in iterations. Students have learned that with boolean variables they manage to execute certain sequences of instructions or not, and use that knowledge without adapting it to different situations [30]. This is very clear in the case of Student 1 and Student 3, who in principle do not use the boolean variable in the while condition but end up using it to correct errors.

The study ends with the execution of the program for several cases and with some reflection questions about the problem and the solution (the program), with the aim of consolidating formal knowledge and accounting for the *trans* stage in its construction. All the students participating in the study finally reached a correct program. Some of them did it in fewer versions and others needed to write more versions. Also, other types of errors appeared in the versions, such as syntax errors, confusion between index and cell value, etc., which are not discussed here, but which were also corrected by the students.

5 Summary and further work

In this paper a research model in didactics of programming is presented. The model has been built over several years of investigating the learning of different basic algorithms and of the corresponding programs execution. The investigations encompass empirical studies designed according to principles of Piaget's epistemological theory. The studies were carried out using a methodology based on Piaget's clinical method used in his studies on genetic psychology. The questions of the research interviews are related to ways of helping students to use their cognitive resources to attain higher levels of knowledge.

One of the most important theoretical ideas about knowledge construction is the categorization of different stages, called by Piaget and Garcia in [31] as the triad *intra-inter-trans*. Accordingly, the research done by the authors of this paper on the construction of concepts focuses on how students transform their instrumental knowledge (*intra* stage) into conceptualized knowledge (*inter* stage) and how this becomes an academic formalization (*trans* stage). The main points of the passage from one stage to another are:

- students succeed in conceptualizing their know-how (*intra* stage), by means of reflecting about how they solve a problem and why the solution works, especially when they themselves experience the need for a base case (or several). The evidence of the passage is given by students' correct descriptions in natural language of their algorithmic solutions and of the reasons for their success (*inter* stage) (section 3.2).

- introducing a formal language as a new object about which the students have to construct knowledge using the same theoretical principles. That means starting the interaction with the elements of the formalism, allowing students to work with particular cases of formal expressions. Besides, by means of introducing automatization, students' thought detaches from particular cases and makes the leap to general solutions, characteristic of the *trans* stage (Section 4.2).

The passage from *intra* to *inter* stage is regulated by Piaget's general law of cognition [28], describing the algorithmic thinking. In the case of programming, where the construction of knowledge is about *programs*, new problems arise due to programs dual nature [38,41]. As is often the case, the need to extend the theory became visible in practice; during an empirical study [39]. One of the objectives of said study was to make the students aware of the causal relationship between their actions and the events in the computer. The results leaded to an extension of Piaget's general law of cognition, constructing the theoretical concept corresponding to the passage from a human being solving a problem to a human being instructing a computer to solve a problem [39,40]. This extension of the general law of cognition (section 4.1) regulates the passage from the *inter* to *trans* stage for the case of knowledge about *programs* as explained in the analysis of the three cases in section 4.2.

In summary, the paper describes a research model on students' construction of knowledge about programs. The process starts from students' informal knowledge (they solve a problem, *intra* stage), passing by the stage of conceptualizing that knowledge by means of writing their solutions (algorithms, *inter* stage), toward the stage of transforming their concepts in formal objects (programs, *trans* stage). The process is regulated by the extended general law of cognition whose diagram in section 4.1 describes the correspondence between algorithmic thinking (first row of the diagram) and the so called computational thinking (second row). The model also accounts for the cognitive mechanisms and tools involved in the construction, represented by the arrows and braces in the diagram.

This model constitutes a contribution to developing CSE or Didactics of Informatics -specifically of programming- as a scientific discipline in the field of CS or Informatics. Like any other area of CS, didactics of programming is a discipline with its own theories and methodologies of research, different from experiences based on opinion issues, as described in section 1.2 and explained in [1].

Some lines of research are relevant as further work:

On the one hand, to study the construction of classes of algorithms and programs, to complete the study of the *trans* stage. This includes for instance, other sorting/searching/counting/etc algorithms that could be compared with each other; recursive or iterative implementations in different programming languages, including those of the functional paradigm as Haskell. More empirical studies about the construction of knowledge on programs are also needed. The analysis of their results reinforces theoretical constructions as is the case of the meaning of computational thinking of the extended general law of cognition (section 4.1), and the analysis of the results of the study of linear search described in section 4.2.

On the other hand, the critical historical analysis has to be included in the investigations since this analysis teaches about the construction of knowledge and can therefore cast light on the learning process [31]. Especially relevant is the historical evolution of concepts of iteration, induction and recursion that are the higher forms of formalized knowledge of ways of solving problems by repeating actions. Also, the historical development of programming languages is undoubtedly an indispensable study for the formalization stage.

Finally, to elaborate pedagogical proposals and didactic guidelines based on this model, can be useful in supporting teachers' researches and practices, and a way of getting institutional support, as well. Especially, giving clear definitions and theoretical meaning to terms and expressions - for instance to *computational thinking* - helps the teachers in attaining Pedagogical Content Knowledge.

References

[1] C. Holmboe, L. McIver, C. George, "Research Agenda for Computer Science Education" (2001) in G.Kadoda (Ed). Proc. PPIG 13, pp 207-223, 2001.

[2] Computing at School (CAS) Web site. Available from https://www.computingatschool.org.uk/

[3] CODE Web site. Available from https://code.org/

[4] Computer Science Teachers Association (CSTA) Web site. Available from https://www.csteachers.org/

[5] Enseignement Public et Informatique. https://www.epi.asso.fr/

[6] S. Papert, Mindstorms: Children, Computers, and Powerful Ideas (1980). Basic Books, New York.

[7] G. Dowek, "Quelle informatique enseigner au lycée? " Bulletin de l'APMEP, (2005) Available from https://www.apmep.fr/Quelle-informatique-enseigner-au

[8] P. Hubwieser, M. Armoni, M. Giannakos, R. Mittermeir, "Perspectives and Visions of Computer Science Education in Primary and Secondary (K-12) Schools" (2014). Journal ACM Transactions on Computing Education (TOCE), Special Issue on Computing Education in (K-12) Schools. Volume 14 Issue 2, June 2014, Article No. 7.

[9] N. Brown, M. Kölling, T. Crick, S. Peyton Jones, S. Humphreys, S. Sentence, "Bringing Computer Science Back into Schools: Lessons from the UK" (2013). Proceeding SIGCSE '13, Proceeding of the 44th ACM technical symposium on Computer science education, pp 269-274.

[10] L. Shulman, "Those who understand: knowledge growth in teaching" (1986). Educational Researcher, 15, pp 4-14.

[11] M. Saeli, J. Perrenet, W. Jochems, B. Zwaneveld, "Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective" (2011). Informatics in Education, 2011, Vol. 10, No. 1, pp 73-88.

[12] A. Schwill, "Computer Science Education Based on Fundamental Ideas" (1997). Proceedings of the IFIP TC3 WG3.1/3.5 joint working conference on Information technology: supporting change through teacher education, pp 285-291.

[13] J. Wing, "Computational thinking and thinking about computing" (2008). Phil. Trans. R. Soc. A(2008)366, pp 3717–3725, doi:10.1098/rsta.2008.0118.

[14] C. E. George, Experiences with Novices: "The importance of Graphical Representations in Supporting Mental Models" (2000). In A.F.Blackwell and E.Bilotta (Eds). Proc. PPIG 12, pp 33-44.

[15] T. Götschi, I. Sanders, "Mental models of recursión" (2003). ACM 1-58113-648-X/03/0002.

[16] T. Budd, "An Active Learning Approach to Teaching the Data Structure Course" (2006). ACM SIGCSE'06, Houston, Texas, USA.

[17] A Gomes, "Learning to program - difficulties and solutions" (2007). International Conference on Engineering Education ICEE.

[18] L. Mannila, M. Peltomäkia, "What about a simple language? Analyzing the difficulties in learning to program" (2007). International Conference on Engineering Education ICEE 2007, pp 211-227.

[19] J. Moström, "A Study of Student Problems in Learning to Program" (2011). Department of Computing Science Ume University, SE-901 87 Ume, Sweden, ISBN 978-91-7459-293-1.

[20] C. Lewis, Exploring Variation in Students' "Correct Traces of Linear Recursion" (2014). Proceedings of the 10th Annual conference on International computing education research (ICER) Pp 67-74.

[21] K. Stephens-Martinez, A. Ju, K. Parashar, R. Ongowarsito, N. Jain, S. Venkat, A. Fox, "Taking Advantage of Scale by Analyzing Frequent Constructed-Response, Code Tracing Wrong Answers" (2017). Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER) pp 56-64).

[22] R. Lister, "Concrete and other neo-Piagetian forms of reasoning in the novice programmer" (2011). Proceedings of ACE 2011 : The 13th Australasian Computing Education Conference Volume 114 pp 9-18, Perth, Australia January 17-20, 2011.

[23] D. Teague, "Neo-Piagetian Theory and the Novice Programmer". Proceedings of the 25th Annual Psichology of Programming Interest Group, University of Sussex, UK, 2014.

[24] G. Dowek, "Les quatre concepts de l'informatique" EPI web site. Available from https://www.epi.asso.fr/revue/articles/a1204g.htm

[25] P. Andrews, "Conditions for learning: a footnote on pedagogy and didactics" (2007), ATM Available from https://www.atm.org.uk/write/MediaUploads/Journals/MT204/Non-Member/ATM-MT204-22-22.pdf

[26] J. Piaget, Genetic Epistemology, (a series of lectures delivered by Piaget at Columbia University, translated by

Eleanor Duckworth) (1968) Available from https://www.marxists.org/reference/subject/philosophy/works/fr/piaget.htm

[27] J. Piaget & coll, *La Formation des Raisonnements Recurrentiels* (1963). France: Presses Universitaires de France.

[28] J. Piaget, *La prise de conscience* (1964). France: Presses Universitaires de France.

[29] J. Piaget, *Success and Understanding* (1978). Harvard: Harvard University Press.

[30] J. Piaget, *Recherches sur la Generalisation* (1978). France: Presses Universitaires de France.

[31] J. Piaget, R. García, *Psychogenesis and the History of Sciences* (1980). New York: Columbia University Press.

[32] G. Brousseau, *Iniciación al estudio de la teoría de las situaciones didácticas* (2007). Translated by D. Fregona. Argentina: Zorzal.

[33] S. da Rosa, "Designing Algorithms in High School Mathematics" (2004). Lecture Notes in Computer Science, vol. 3294, Springer-Verlag.

[34] S. da Rosa, "The Learning of Recursive Algorithms from a Psychogenetic Perspective" (2007). Proceedings of the 19th Annual Psychology of Programming Interest Group Workshop, Joensuu, Finland, 201-215.

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

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

[37] S. da Rosa, "The construction of knowledge of basic algorithms and data structures" (2015) Proceedings of the 26th Annual Psychology of Programming Interest Group Workshop (PPIG2015), Bournemouth, United Kingdom, Jul 2015.

[38] S. da Rosa, "Studying preconceptions of novice learners about program execution" (2016) Proceedings of the 27th Psychology of Programming Interest Group Workshop, Cambridge, UK 2016.

[39] S. da Rosa, A. Aguirre, "Students teach a computer how to play a game" (2018). Lecture Notes in Computer Science, vol. 11169, Springer-Verlag.

[40] S. da Rosa, "Piaget and computacional thinking" (2018). Proceedings of CSERC 2018 : The 7th Computer Science Education Research Conference Saint-Petersburg Electrotechnical University "LETI", Saint-Petersburg, Russia October 10-12, 2018. In ACM Digital Library.

[41] A. Chmiel, S. da Rosa, F. Gómez, "Philosophy of Computer Science and its Effect on Education - Towards the Construction of an Interdisciplinary Group", (2013). Clei Electronical Journal Volume 19 : Number 1 : Paper 5, 2016.