Designing a didactic model for programs and data structures

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
Several authors affirm with solid arguments that it is essential to educate in computing, at least from secondary education and covering undergraduate courses, and that this continues to be a pending problem in most educational systems. Some point to the relationship between research and educational practice, which partly arises from the undervalued role of didactic research within the academy. Based on our epistemological model and taking as starting point fundamental ideas of computing, we began to develop a didactic model for the development of computational competencies and skills for novice students. In this paper we present the rationale of the proposed didactic model, a description of and empirical study and a preliminary analysis of the results of the experience.

1. Introduction
Several authors affirm with solid arguments that it is essential to educate in computing, at least from secondary education and covering undergraduate courses. These arguments provide answers to the *why* and *for whom* (to teach computer science) of the didactic questions (Saeli, Perrenet, Jochems, & Zwaneveld, 2011). The authors add that this continues to be a pending problem in most educational systems (Denning & Tedre, 2015, 2019, 2021; Dowek, 2013) and some point to the relationship between research and educational practice, which partly arises from the undervalued role of didactic research within the academy. For example, at the conference "Key Competencies in Informatics and ICT (KEYCIT 2014)" that took place at the University of Potsdam in Germany in 2014[1], several works by science educators from various European countries were presented. In those, case studies, positions and perspectives of education in computing and technology were discussed, focused on secondary education, undergraduate education and teacher training. We found that the concepts of "competencies" and "key competencies" are a central issue and that most authors use some type of taxonomy to define their didactic model, for example in (Bröker, Kastens, & Magenheim, 2014) the authors take the following definition of competency: "The existence of learnable cognitive abilities and skills which are needed for problem solving as well as the associated motivational, volitional and social capabilities and skills which are essential for successful and responsible problem solving in variable situations." and they add: "This definition implies that competences are learnable by interventions." For the competency model, these authors use the Anderson and Krathwohl taxonomy (AKT), an adaptation of Bloom's taxonomy to which they add two dimensions: A) levels of knowledge (factual, procedural, conceptual, metacognitive) and B) classification of cognitive domains ("remembering, understanding, applying, analyzing, evaluating, creating").

As a result of the literature review, added to our empirical research and own theoretical development, we conclude that our model of knowledge construction about data structures, algorithms and programs, helps in designing answers to the *how* to teach of the didactic questions (Saeli et al., 2011), playing a role similar to that of competencies and taxonomies used by the reviewed authors. Besides, the model contributes with a theoretical elaboration, mainly in two directions: extending Piaget's theory to encompass the construction of knowledge about programs (da Rosa, 2018; da Rosa, Viera, & García-Garland, 2020) and providing a theoretical framework for designing empirical studies through Piaget's triad intra-inter-trans (da Rosa & Gómez, 2019, 2022). This contribution is not minor: in their systematic review of research in computer science education, several of the cited authors found that half of the studies do not explain any theoretical framework. For the studies that do, their theories and conceptual models

[1]https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/7032/file/cid07.pdf

are taken from other areas such as psychology or pedagogy and present a dispersed area with a great variety of terminology and methods. Although the authors anticipate a growth in the theoretical field of computer science education, at the time of their study they considered the number of studies with theoretical and conceptual frameworks specific to the area so small that they would not have sufficient impact to generate a theoretical unification of the area (Malmi et al., 2014).

Based on our epistemological model and taking fundamental ideas of computing (Schwill, 1997; Bell, Tymann, & Yehudai, 2018) as a didactic perspective, we began to design a didactic model for the development of computational competencies and skills for novice students (Cabezas & da Rosa, 2022). Fundamental ideas group together the central concepts and long-range of computing, allowing knowledge to be distinguished from ephemeral information, which constitutes a suitable answer for the *what* to teach of the didactic questions. These fundamental ideas are described in the next Section. The rest of the paper is organized as follows: in Section 3 a complete empirical study is included and in Section 4 some reflections and future lines of work are presented. Finally, bibliographic references are included.

2. Fundamental ideas of computing

Andreas Schwill's work on fundamental ideas in computer science (Schwill, 1997) is a classic cited by several authors as a starting point for the development of didactic modeling. Schwill defines four criteria that a fundamental idea must meet:

- the vertical criterion (the idea appears in different domains of the discipline)
- the horizontal criterion (the idea can be worked on at any intellectual level)
- the criterion of time (the idea can be observed throughout the evolution of the discipline)
- the common sense criterion (the idea makes sense in an informal, pre-theoretical and pre-scientific context and can be expressed in natural language)

and points out algorithmization, structural dissection and language as fundamental ideas (with subideas). In (Dowek, 2012) the author adds as fundamental the idea of a machine related to the notion of a program as an executable object. In (Bell et al., 2018), the authors propose ten fundamental ideas, listed below, that cover those of Schwill and Dowek and add others related to the further development of computer science (networking, security, simulations).

1. Information is represented in digital form.
2. Algorithms interact with data to solve computational problems.
3. The performance of algorithms can be modelled and evaluated.
4. Some computational problems cannot be solved by algorithms.
5. Programs express algorithms and data in a form that can be implemented on a computer.
6. Digital systems are designed by humans to serve human needs.
7. Digital systems create virtual representations of natural and artificial phenomena.
8. Protecting data and system resources is critical in digital systems.
9. Time dependent operations in digital systems must be coordinated.
10. Digital systems communicate with each other using protocols.

The specificity of the modeling process means that we have focused on building knowledge about algorithms, data structures, and programs. Consequently, we face the challenge of defining our own subset of fundamental ideas. In principle we take the fundamental ideas above related to these concepts (1, 2, 3, 4, 5), separating them from those related to issues such as security, networks and ethics (the rest).

The authors' formulation of ideas 2, 4 and 5 led us to investigate the notions of a non-computable problem, problems without a computable solution and a non-computational problem since their clarity is relevant for our subset of fundamental ideas. Thus we find that in the complementary document to their article[2], the authors delve into each of the ten fundamental ideas proposed, expressing about idea 2: "The term 'computational problem', 'algorithmic problem', or simply 'problem' in this context is often used to refer to the task that needs to be computed e.g. searching for a word, sorting values into order, finding the shortest route on a map, or finding a face in a photo." At this point the importance of our theoretical elaboration is clearly observed: in our epistemological model the "algorithmic world" is distinguished from the "computational world" and we point out that "algorithmic problem" and "computational problem" are not the same (and much less simply "problem"!), even in a computer context. One of the main contribution of our work is the extension of Piaget's general law of cognition to encompass the construction of knowledge about data types and programs. Piaget's original law regulates the construction of knowledge about *algorithmic solutions* of problems and we have extended it to take into account *computational solutions*. The construction of knowledge about algorithms, data types and programs lies in understanding the dialectical relationship between both kind of solutions (da Rosa et al., 2020). In our work fundamental idea 2 is related to the "algorithmic world" and fundamental idea 5 to the "computational world".

In (Harel & Feldman, 2004) the author defines **algorithmic problem** as:

1. a characterization of a legal, possibly infinite collection of potential input sets,
2. a specification of the desired outputs as a function of the inputs.

Consequently, the fundamental ideas for which we plan to design didactic sequences and validate them in the classroom are the following:

1. Information is represented in digital form.
2. Algorithms interact with data to solve algorithmic problems.
3. Programs express algorithms and data in a form that can be implemented on a computer.
4. The performance of algorithms can be modelled and evaluated.

The fundamental idea 1 is related to the computational skill of knowing how to represent information as data types that the program has to deal with and here is used in that sense. Designing an algorithmic solution (algorithm) to an algorithmic problem consists of defining a function that takes elements from an input set and produces a desired output (fundamental idea 2), and solving the computational problem means to implement the algorithm in some programming language and execute it (fundamental idea 3). We consider that the fundamental idea 4 has to be introduced also, although it is not included in the empirical study presented in the following Section. As mentioned in the Section 1, the didactic sequence was designed and developed within the theoretical framework of the intra-inter-trans triad.

3. The empirical study

The activities described in this section were developed in two instructional instances with students in an introductory programming course in October 2023. A didactic sequence was designed to introduce

[2]https://www.canterbury.ac.nz/media/documents/oexp-engineering/BigIdeas-webdocument.pdf

a topic that until that moment was new to the students but related with their previous knowledge. It is a data structure called ***capped array***, along with four fundamental operations for its manipulation: *initialization*, *insertion*, *listing* and *search*. In the capped array values are inserted one by one from zero (empty capped array) until the predefined maximum value, without being necessary to store values for a fixed number of cells determined prior to execution as in classical arrays known by the students. The cap is a special variable that keeps track of the number of values stored so far. This structure holds didactic interest in two senses: first, due to its flexibility it makes possible to establish a relationship, in terms of the process of knowledge construction about data structures, between static structures previously studied (*arrays* and *records*) and dynamic structures to be studied later (*linked lists* and *binary trees*). Second, the representation of data through the structure of capped array and operations on it present an adequate level of complexity, enhancing the introduction of fundamental ideas 1, 2 and 3 of Section 2.

The sequence was designed to be executed in class with a group of 12 students, taking into account their prior knowledge. They all had worked with the following topics: resolution of simple programming problems, basic syntax of an imperative programming language, variables, elementary data types, expressions and simple instructions, control structures (both selection and iteration), subprograms (both functions and procedures), static data structures (arrays and records). The programming language used in class was Pascal.

The activities for the sequence were designed based on guidelines established in the epistemological model according to which knowledge is built through a first stage focused on isolated objects (*intra* stage), then passing through a second stage that takes into account relationships between said objects and their transformations (*inter* stage) and reaching a third stage in which a general scheme is built that involves both the generalized objects and their transformations (*trans* stage). The sequence consists of four groups of activities introducing the fundamental ideas 1, 2 and 3 as shown below:

A. Manipulation of the structure (fundamental ideas 1 and 2).

B. Formalization of the structure in a programming language (fundamental idea 1).

C. Initialization and insertion operations (fundamental idea 3).

D. Listing and search operations (fundamental idea 3).

The activities within group A start from the students' instrumental knowledge (*intra* stage) and induce its transformation into conceptual knowledge (*inter* stage) in relation to the structure itself and the insertion operation, both of which constitute new concepts for the students. The activities in this group introduce fundamental ideas 1 and 2 by means of transforming the data in a structure that can be handled by the algorithm in Activity 3. The activities within group B introduce fundamental ideas 1 and 3 by means of discussing computational issues of the representation of the structure and its effects on the memory of the computer. The activities in group C introduce fundamental idea 3 with emphasis in the Pascal program as a formalization of the conceptual knowledge about the insertion algorithm (*trans* stage). Finally, in the activities within group D students work with operations that present similarities and differences with the insertion operation in order to consolidate the knowledge about the new structure of capped array. These are listing and search operations that students have implemented on classic arrays. We present below a detailed description of the activities within each group.

Group A: Manipulation of the structure (fundamental ideas 1 and 2)

Activity 1: *Imagine there is a shelf that contains compact discs, which are located from left to right. It is desired that a person, standing in front of the shelf, can immediately know the number of discs placed so far, without having to count them one by one or do any type of calculation. What could be added to the shelf so that the person can know that?*

The proposed arrangement has a correlation with the data structure to be constructed at the formal stage. The discs correspond to the values to be stored in the cells and their positions on the shelf to the indices of the array (the data structure). The activity introduces the need to incorporate a new element: the *cap*. Students are induced to propose a solution, at instrumental level, to solve the task. They are expected to come up with ideas such as "*put a mark*" or "*write down the amount of discs on a piece of paper*". Based on the students' ideas, they are then asked to reflect on why each alternative works (or doesn't work). For example, putting a mark implies the need to count the discs that precede it, which does not satisfy the requirement of avoiding counting them. Having a piece of paper with the amount provides a better solution, as it avoids having to count all of the discs every time.

Activity 2: *Suppose you have a disc in your hand and you are about to place it on the shelf after the last one. What condition should be met to be able to perform the task successfully? After placing the disc, what should be done so that the person from Activity 1 still knows how many discs are placed without having to count them?*

This activity expects students to apply instrumental knowledge for its resolution. Everyone has stored items on a shelf before, so they should be able to figure it out without difficulty. The purpose is to make them aware of both the general case and the edge case (checking that the cap value does not exceed the maximum number of discs that fit on the shelf).

Activity 3: *Based on your answer to the question in activity 2, write an algorithm in pseudocode to insert a new disc on the shelf after the last one and update the number of discs placed.*

This activity proposes the use of an intermediate formalism (pseudocode) to help conceptualization and begin the passage to the trans stage. This is particularly helpful when introducing a new operation (insertion of a new element) that has no similar equivalent in the students' prior formal knowledge. So far, they had only worked with classic arrays (without a cap). In a classic array, the insertion operation is not defined, since all of its cells always have a value stored in it. The notion of inserting a new element into a data structure is being worked on for the first time with the introduction of the capped array.

Group B: Formalization of the structure in a programming language (fundamental idea 1)

Activity 4: *Define a data type in Pascal that allows representing the shelf along with the number of discs stored so far (the cap). For simplicity, assume that the shelf has the capacity to hold at most 50 discs and that each disc is simply represented by an integer number (its ISBN). Explain how each part of your Pascal definition corresponds to the shelf.*

This activity involves construction of knowledge about the use of the programming language to define the capped array. The students had previously worked separately with arrays and records, and their integration allows the new data structure to be defined, since it unifies, in the same syntactic unit, both the array and the cap. Taking advantage of the fact that no new syntactic elements are required, the activity encourages students to use what they already know about the language syntax to propose a definition for the new data structure and establish a correspondence between the shelf (instrumental stage) and its representation in a programming language (formal stage).

Activity 5: *Given the following variables in Pascal:*

```
arr: Arreglo;        (* classic array *)
act: ArregloConTope; (* capped array *)
```

Assuming that both of them have 10 cells (with indices ranging from 1 to 10), draw both the variable ***arr*** *with values stored in all of its cells and the variable* ***act****, assuming that only the first four cells have been loaded with values so far. What expression should you write in Pascal to access the third cell of the classic array? And to access the third cell of the capped array? And to access the cap?*

Activity 6: *Now we are going to compare some characteristics between the classic array and the capped array. What syntax similarities and differences do they present? In what circumstances is it more appropriate to work with the classic array? and with the capped array? What is the purpose of the cap? Why does the classic array not need a cap? What happens with the values in the cells after the cap in* *the capped array?*

Activity 5 induces students to construct knowledge about the syntax rules necessary to manipulate the capped array and become aware of how said data structure works in the computer memory. Subsequently, activity 6 induces a reflection process about the similarities and differences between the two data structures, both at syntactic and functional level. Especially in relation to the fact that, as seen earlier in the course, all cells of the classic array must be stored with valid values, while the cells after the cap contain undefined values ("garbage" values), simulating the absence of discs on the shelf beyond the position indicated by the cap. According to the epistemological model, the construction of knowledge about the new structure is carried out from the generalization of knowledge previously constructed for the manipulation of classic arrays and records.

Group C: Initialization and insertion operations (fundamental idea 3)

Activity 7: *Write the following subprograms in Pascal* (each suprogram name is kept in Spanish, which is the native language of the students who participated in the class, as well as the names given in Spanish for the data types: *Arreglo* for the classic array and *ArregloConTope* for the capped array):

```
procedure InicializarTope (var act: ArregloConTope)
function EstaLleno (act: ArregloConTope) : boolean
procedure Insertar (val: integer; var act:ArregloConTope)
```

which, respectively, initialize the cap with 0, determine whether the capped array already contains values in all its cells and insert the new value into the capped array, according to the algorithm in activity 3.

Activity 8: *Draw a capped array that has 10 cells in total and show what it would look like after each of the following instructions is executed. How many more values could be inserted into it after executing these instructions?*

```
InicializarTope (act);
Insertar (5, act);
Insertar (3, act);
Insertar (8, act);
```

Activity 7 asks students to implement the requested subprograms in Pascal, based on the knowledge built in the previous activities. It is expected that no major difficulties should arise, due to the progressive construction of concepts arising from previous activities. In particular, the third procedure constitutes a new expression of the algorithm in pseudocode. During writing, it is expected that students will make errors typical of machine execution. For example, a problem with an index that produces an out of range error, or accessing a cell with an undefined value ("garbage" value). In such cases, activity 8 induces students to resort to a mechanism called automation, described in (da Rosa & Gómez, 2022), which consists of manually executing the algorithms on an array drawn on paper which, within the framework of the epistemological model, has proven to be very useful in helping students become aware of errors and return to the code and correct them.

Group D: Listing and search operations (fundamental idea 3)

Activity 9: *Write the following subprograms in Pascal*:

```
procedure ListadoComun (arr: Arreglo)
procedure ListadoConTope (act: ArregloConTope)
```

which respectively list on screen the values stored in the classic array and in the capped array. Watch the code written for both subprograms. What similarities and differences do they present?

Activity 10: *Write the following subprograms in Pascal*:

```
function BusquedaComun (arr: Arreglo; num: integer): boolean
function BusquedaConTope(arr: ArregloConTope; num:integer):boolean
```

which respectively determine whether or not the value num is stored in the classic array and in the capped array. Watch the code written for both subprograms. What similarities and differences do they present?

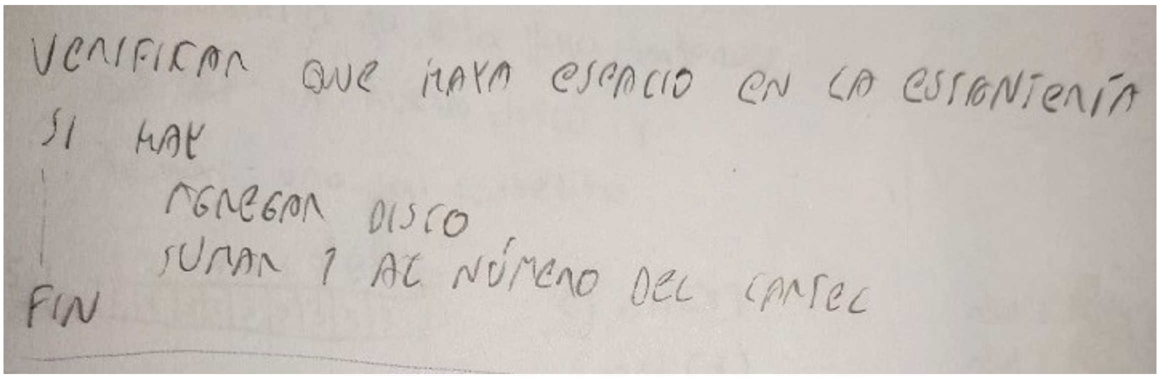
Unlike the insertion algorithm from activity 8 (which is a *new* operation), students have already implemented listing and search operations on a classic array earlier in the course. These activities take this into account and seek to adapt said prior knowledge for the implementation of new versions, now working on the capped array. They have reasonable similarities and differences with the classic array, which provides an adequate context for the construction of new knowledge working at the formal level (through generalization). Finally, students are asked to compile and run all implemented subprograms on a computer. To do this, they are provided with a source file with the declarations and headers of the requested subprograms. Students simply must complete the missing portions, corresponding to the body of each of the implemented subprograms, before compiling and executing.

The course lasts 15 weeks and has several groups of students, distributed at different time schedules. Every group attends two classes per week, each lasting 90 minutes. According to the course curriculum, the capped array is planned to be worked on in week 10, after having introduced classic arrays and records in weeks 8 and 9, respectively. The didactic sequence was executed with a group of 12 students, who participated voluntarily. The rest of the students attended groups in which the topic was worked on in the traditional way (presenting from the beginning both the data structure and the operations, working directly at the formal stage), without taking into account the students' prior knowledge at the *intra* and *inter* stages. A future in-depth analysis of the differences in learning observed in the students after having worked on the topic based on one methodology or another is to be done.

All twelve students worked in teams of three members each. Everyone solved the ten activities presented. The work mechanics consisted of the teacher presenting the statement of each activity and then each team proposed a solution for it. Subsequently, a collective discussion was held among the entire class. We now present the development of the activities within each group (A, B, C and D) and a brief analysis of the work of the different teams.

Group A: Manipulation of the structure (fundamental ideas 1 and 2)

In activity 1, the four teams determined necessary to have a "space" that kept track of the number of discs placed so far on the shelf. They called it "poster", "sheet" or "paper". Regarding the question posed in activity 2, everyone concluded that it was necessary to compare the value currently noted on the "space" with the capacity of the shelf prior to placing the new disc after the last one. Within the framework of the epistemological model, students became aware of the two fundamental operations to solve the problem: *comparison* (of the cap with the maximum size) and *insertion* (of the new element after the last one) and they all managed to write the requested pseudo code in activity 3. As an example, the solution of one team is shown in Figure 1 (photo of the original algorithm, written in Spanish).

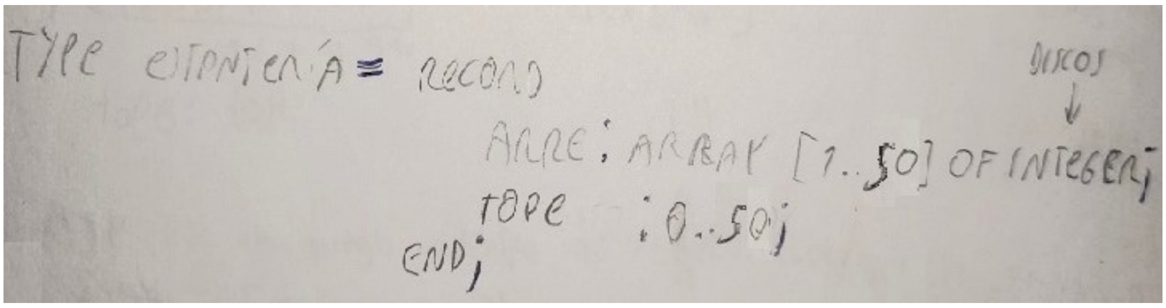


```
VERIFICAR QUE HAYA ESPACIO EN LA ESTANTERÍA
SI HAY
    AGREGAR DISCO
    SUMAR 1 AL NÚMERO DEL CARTEL
FIN
```

Figure 1 – Pseudo code for activity 3

Group B: Formalization of the structure in a programming language (fundamental idea 1)

For activity 4, all teams defined a *record* type in Pascal that grouped the array with the cap. As expected, they did it tied to the specific instance presented in the activities of group A, considering a shelf with capacity for 50 discs. As an example, the definition proposed by one team is shown in Figure 2 ("Estantería" is the Spanish word for shelf). According to the epistemological model, such definitions



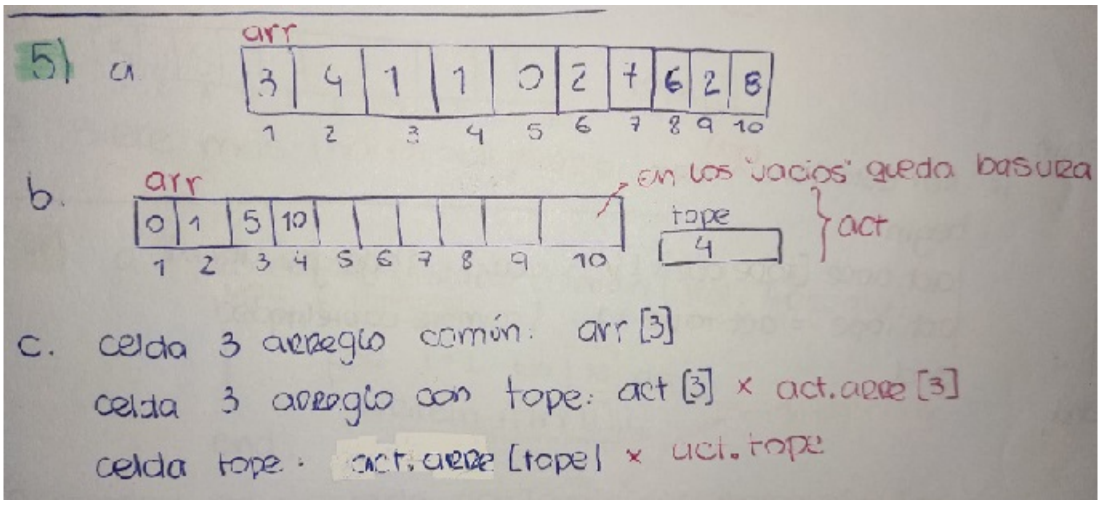
```
TYPE ESTANTERÍA = RECORD                          DISCOS
                                                     ↓
                   ARRE: ARRAY [1..50] OF INTEGER;
              TOPE    : 0..50;
       END;
```

Figure 2 – Type definition for the capped array

are expected in this stage. Although they are correct from a syntactic point of view, they show that the students' thinking is still tied to the specific instance of discs manipulated in the instrumental stage. The jump from specific cases to the general case is produced by successive repetition. To abstract from the specific instance, during the collective discussion, the teams were asked how they would modify the definition so that it serves different specific sizes. This introduced the need for a constant N to make it possible to define a capped array of any size, as it is shown below:

```
type ArregloConTope = record
          arre: array [1..N] of integer;
          tope: 0..N;
end;
```

In activity 5, the four teams managed to draw both the classic and the capped array, according to the instructions given. When writing the expressions to access the third cell of the classic array, the third cell of the capped array, and the cap, they made various syntax errors. This is expected at this stage, since it is the first time that they combine syntax for arrays with syntax for records. In the subsequent discussion, reflection was induced and all teams were able to correct the aforementioned errors. As an example, the responses of one of the teams are shown in Figure 3.



c. celda 3 arreglo común: arr[3]
celda 3 arreglo con tope: act[3] × act.arre[3]
celda tope: act.arre[tope] × act.tope

Figure 3 – Answers for questions posed in activity 5

As in activity 5, the answers to the questions posed in activity 6 presented a variety of errors, which were again corrected after discussion, in which the similarities and differences between both structures (classic and capped array) were stated, both in terms of syntax and semantics. All students understood that it is appropriate to use the classic array when the number of values is fixed and known in advance, while the capped array is a collection with a variable number of elements, upper bounded by the size of the array.

Group C: Initialization and insertion operations (fundamental idea 3)

In activity 7, the four teams wrote initial versions for the three requested subprograms, making similar errors to those in activity 6. The syntax errors became evident after computer compilation. Facing the errors shown by the compiler helped the teams to reflect and correct them. Regarding execution errors (for example: an out-of-range error), the automation mechanism (guided by activity 8) made it possible to detect and correct them. The final version of one team for each requested subprogram is shown below:

```
procedure InicializarTope (var act: ArregloConTope);
begin
    act.tope:=0;
end;

function EstaLleno (act: ArregloConTope) : boolean;
begin
    if act.tope=N then
        EstaLleno:=TRUE
    else EstaLleno:=FALSE;
end;

procedure Insertar (val: integer; var act: ArregloConTope);
begin
    if (not EstaLleno(act)) then
    begin
        act.arre[act.tope+1]:=val;
        act.tope:=act.tope+1;
    end;
end;
```

Group D: Listing and search operations (fundamental idea 3)

Finally, activities 9 and 10 had a development analogous to that of activities 7 and 8, once again making it possible for all teams to implement correct versions for the requested subprograms (listing and search). Each team compiled and then ran all implemented subprograms on the computer. Below is the final version corresponding to the same team of questions 7 and 8 for the listing and search on the capped array.

```
procedure ListarConTope (act: ArregloConTope);
var i: integer;
begin
    for i:= 1 to act.tope do
        writeln (act.arre[i]);
end;

function BuscarConTope (act: ArregloConTope; num : integer) : boolean;
var i:integer;
begin
    i:= 1;
    while (i <= act.tope) and (act.arre[i] <> num) do
```

```
        i:= i+1;
    if i <= act.tope then
        BuscarConTope:=TRUE
    else BuscarConTope:=FALSE;
end;
```

4. Conclusions and further work

As it is pointed out in Section 1, various authors argument that one of the difficulties for computing being part of the basic disciplines for the education of all students lies in the undervalued role of its didactic. This becomes evident considering the lack of scientific research in the field supported with solid theoretical foundations elaborated with the participation of computing teachers. We have conducted for many years, empirical studies such as the one presented here, designed in the framework of epistemological basis. In the last years we have integrated results from didactic research (Bröker et al., 2014; Schwill, 1997; Bell et al., 2018) in order to produce a didactic model that computing teachers can use and evaluate. Particularly we found that the fundamental ideas of (Schwill, 1997; Bell et al., 2018) constitute an axis for designing didactic sequences, as described in Section 3. Although we have included the fundamental idea 4 about the performance of algorithms in our subset of fundamental ideas (see Section 2), we have not used it in the empirical study described here, leaving the task for future work.

The didactic sequence designed for the study constitutes one of the applications of the didactic model (didactic sequences for Physics are described in a draft version of unpublished manuscript). The theoretical framework guided by the fundamental ideas and the epistemological model that we have developed (Section 2) made it possible to design the activities in such a way that both the learning objectives and the work strategies were clearly outlined.

Within the four fundamental didactic questions: *what*, *how*, *why* and *for whom* (to teach computer science) (Saeli et al., 2011), the prior identification of the fundamental ideas made it possible to accurately define the specific programming concepts to work on in the sequence (*what*). On the other hand, the epistemological model made it possible to define the activities so that the students themselves built knowledge about these concepts during their execution (*how*). We believe that this particular point is of special value and distinguishes our didactic model from other approaches traditionally used in education, which usually focus on the presentation of concepts, without taking into account how students learn. As for *why* and *for whom*, the authors mentioned in Section 1 solidly justify why it is important to educate on computer topics in secondary education and/or undergraduate courses.

Last, in relation to the application of the didactic sequence, it was observed, in a first preliminary analysis, that the gradual nature of the proposed activities (taking into account the knowledge construction process guided by the *intra-inter-trans* triad) made it possible for the students to gradually build knowledge about the concepts worked on. The four teams of students managed to successfully solve every activity and finished with the compilation and execution on the computer of all the subprograms, implemented by themselves. The teacher's main task was to guide the collective discussion after each activity (instead of having an expository role of the concepts being worked on) and make corrections when appropriate. An in-depth analysis of the study remains to be done, from which it is expected to extract more evidence about the effect of introducing the fundamental ideas and draw conclusions that, in turn, will allow to provide feedback and enrich the didactic model, still under construction.

5. References

Bell, T., Tymann, P., & Yehudai, A. (2018). The big ideas in computer science for k-12 curricula. *Bulletin of EATCS, 1(124)..* `http://smtp.eatcs.org/index.php/beatcs/article/viewFile/521/512`.

Bröker, K., Kastens, U., & Magenheim, J. (2014). Competences of undergraduate computer science students. *In KEYCIT – Key Competencies in Informatics and ICT. Torsten Brinda, Nicholas Reynolds,* *Ralf Romeike, Andreas Schwill (Eds..*

Cabezas, M., & da Rosa, S. (2022). Modelado didático para ideas fundamentales en computación. *Proceedings of The 51 SADIO Conference, Simposio Argentino de Educación en Informática (SAEI 2022)*. `https://www.fing.edu.uy/~darosa/#papers/Modelado-Didactico-Ideas-Fundamentales-Computacion.pdf`.

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., & Gómez, F. (2019). Towards a research model in programming didactics. *Proceedings of 2019 XLV Latin American Computing Conference (CLEI)*, 1–8. doi: 10.1109/CLEI47609.2019

da Rosa, S., & Gómez, F. (2022). The construction of knowledge about programs. *Proceedings of PPIG 2022 - 33rd Annual Workshop*, 1–8.

da Rosa, S., Viera, M., & García-Garland, J. (2020). A case of teaching practice founded on a theoretical model. *Lecture Notes in Computer Science 12518 from proceedings of the International Conference on Informatics in School: Situation, Evaluation, Problems*, 146–157. `https://doi.org/10.1007/978-3-030-63212-0`.

Denning, P., & Tedre, M. (2015). *Shifting identities in computing: From a useful tool to a new method and theory of science*. In Hannes Werthner and Frank van Harmelen, Eds. Informatics in the Future, Proceedings of the 11th European Computer Science Summit.

Denning, P., & Tedre, M. (2019). *Computational thinking*. Cambridge, MA : The MIT Press.

Denning, P., & Tedre, M. (2021). Computational thinking: A disciplinary perspective. *Informatics in Education.*, *20(3)*, 361–390. doi: 10.15388/infedu.2021.21

Dowek, G. (2012). Les quatre concepts de l'informatique. *Bulletin de l'Association EPI*. `https://www.epi.asso.fr/revue/articles/a1204g.htm`.

Dowek, G. (2013). *L'enseignement de l'informatique en France, Il est urgent de ne plus attendre*. `www.academie-sciences.fr/activite/rapport/rads_0513.pdf`. (Rapport de l'Académie des Sciences)

Harel, D., & Feldman, Y. (2004). *Algorithmics - The Spirit of Computing*. Addison-Wesley Publishers Limited 1987, 1992, Pearson Education Limited 2004.

Malmi, L., Sheard, J., Bednarik, B., Helminen, J., Kinnunen, P., Korhonen, A., ... Simon. (2014). Theoretical underpinnings of computing education research: what is the evidence? *ICER14: Proceedings of the tenth annual conference on International computing education research.*, 27–34.

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2011). Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective. *Informatics in Education, Vol. 10, No. 1*, 73–88.

Schwill, A. (1997). Computer science education based on fundamental ideas. *Proceedings of the IFIP TC3 WG3.1/3.5 joint working conference on Information technology: supporting change through teacher education*, 285–291.