Piaget and Computational Thinking

Sylvia da Rosa Zipitría

Institute of Computing, Facultad de Ingeniería, Universidad de la República
darosa@fing.edu.uy

Abstract. In this article I present a theoretical framework for the concept computational thinking. I do so in response to some of the problems and consequences of the lack of viable theoretical foundations; especially in relation to the development in recent years of many educational practices that claim the term computational thinking. I therefore introduce my extension of Jean Piaget's general law of cognition which arose as a result of my empirical research on novice learners knowledge of the concept of a program as an executable object. Said empirical study is briefly described in this paper as a means to highlight the key to my extension of Piaget's general law, which is the insight of how the thought processes and methods involved in cases where the subject must *instruct an action to a computer* differ from those in which the subject instructs another subject, or performs the action themselves. My theory explains the difference between algorithmic thinking and computational thinking by adequately locating it in the specificities of the subject instructing a computer. Hence, in this article I claim that my extension of Piaget's law offers a more empirically thorough and theoretically sound way forward in the conceptual development of computational thinking than the alternatives that are being debated in academia to the present day.

Keywords: Learning to program · Novice learners · Piaget's theory.

1 Introduction

In this article I discuss the concept of computation thinking (hereinafter CT). More specifically, I discuss the recent upsurge of interventions that use the term computation thinking to describe certain ideas and practices in educational settings, as well as the academic debates that these educational ideas and practices have sparked in recent years. My intention is to bring clarity to a concept that is becoming increasingly popular in practice and, while it is intensely discussed and debated, has theoretical contours that remain poorly defined. I argue that the popularity of the concept in educational practice has been producing teaching tools and interventions that, not only obscure the very purpose of applying computational thinking in education, but also fail to produce the desired outcomes in student learning.

CT as it is being debated in recent years is a concept that, among other things, is not easily distinguished from other related concepts such as, for example, algorithmic thinking.

The problem of conceptual vagueness is not a problem in itself. However, certain practices and applications of diffuse concepts can become problematic. Concepts that appear to offer new insights which could potentially be converted into practical tools to be applied in various fields can become immensely popular. There is a risk, however, of a concept being sabotaged by its own popularity as its theoretical development is intersected by its practical application in unforeseen ways. This is particularly true at the interface of policy and research which implies a complex interaction of interests, institutionalised processes and techniques that ultimately produce a wide range of (not always desirable) policy interventions and results.

The case of the introduction of computer science as a discipline in educational policy is an example of the difficulties that can emerge in such contexts. Since the expression "Computational Thinking" was first introduced in 2006 by Jeannette Wing, a review of education literature [6, 4, 25, 2, 21, 5, 3], shows a diversity of approaches, encompassing definitions, interventions and assessments, that continually generate the most diverse opinions and interpretations, but fail to offer a satisfactory conceptual landscape within which such a wide range of ideas and practices can be contained.

CT has been making the transition from computer science concept into educational tool and suffers, like many concepts before it, from a disconnect between its theory and its practical application, making it vulnerable to confusion and misuse. As Denning writes:

The absence of clear definitions and substantiated claims, "... leave teachers in the awkward position of not knowing exactly what they supposed to teach or how to assess whether they are successful." Peter Denning in [4].

In fact, the application of a concept that is theoretically weak can even be counterproductive. As Paulson notes:

"Unless somebody can come up with a more insightful definition, it is indeed time to retire 'computational thinking'". Lawrence C. Paulson in [7].

Indeed, before the promised potential of CT is lost in misguided practice it is the task of researchers (myself included) to map the current theoretical landscape of computational thinking and identify how it can be moved forward.

I therefore begin with a brief literature review of the emergence and trajectory of CT, and the conceptual challenges produced since it has been applied, particularly in education.

More specifically, I review the two perspectives that have been most relevant in the recent trajectory of CT; on the one hand, the literature seeking to define CT using computational models. On the other hand, the literature that define CT based on the cognitive sciences, hence taking psychological processes into consideration. I emphasise these two perspectives in the review because both build on areas that are relevant to the development of CT (i.e. computer and thought).

Other literature, such as papers based on other approaches (for instance CHB or assessing CT), do not significantly add further insight to the analysis and has therefore been excluded from the review.

The paper is organised into the following sections: 1 Introduction; subsection 1.1 CT in the literature; section 2; Algorithmic thinking; subsection 2.1 The general law of cognition; section 3 Extending the general law of cognition: computational thinking; subsection 3.1 The experience prior to the theory; section 4 Conclusions; section 5 Acknowledgements and finally References.

1.1 CT in the literature

The expression "Computational Thinking" was introduced in 2006 by Jeannette Wing, [22], where she characterises CT as: solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science. In her article Wing makes the remark that CT is done by humans, not machines. Since then, many researchers have continued to develop and define the concept, including Wing herself (see later on this section).

In [4], Peter Denning makes a brief historical review from the roots of algorithmic-computational thinking, including references of several computer science researchers like Donald Knuth, Edsger Dijkstra, Seymour Papert, among others. These authors have mentioned in one or another form the way of thinking when solving algorithmic problems and representing their solutions as algorithms, characteristic of computer science. The focus of the majority of authors is on CT as a term meant to encompass a set of concepts and thought processes that aid in formulating problems and their solutions in different fields in a way that could involve computers. In most of the cases, these include; abstraction, recognising patterns, logical reasoning, automation, testing, generalising, data representation and so forth.

In an effort to clarify the specificity of CT, computer science researchers have sought to link CT to a specific computational model; a strategy which they believe could provide CT with a much needed conceptual clarity. Among others, Denning is responding to Alfred Aho's words from [1], where Aho emphasises the relevance of computational models, not only in computer science but also in other disciplines. Denning, in agreement with Aho, points out the absence of any mention of computational models as one of the main sources of ambiguity in the definitions of CT.

Further, Denning argues that the problem lies in failure to realize the fact that computational concepts (abstraction, recognising patterns, logical reasoning, automation, testing, generalising, data representation and so forth) constitute a computational model. Others, however, such as [7], Lawrence C. Paulson downplay the impact of relying to computational models for defining computational thinking; mainly because "All disciplines rely on models".

While many computer science researchers have been debating the issue of whether the problem of the confusion surrounding CT can be addressed by specifying to which computational model does that CT refer, other authors present a different perspective without making references to computational models or even computational concepts.

These authors offer a view much more centred around the cognitive sciences and mental processes. For this reason, much of their debate is concerned with who is the thinker in CT and where does the processing occur.

For instance, Wing, Cuny and Snyder in [24] write: "Informally, computational thinking describes the mental activity in formulating a problem to admit a computational solution. The solution can be carried out by a human or machine, or more generally, by combinations of humans and machines". While developing this thought, Wing has had to make changes to her initial definition in order to include " thinking process where solutions are represented in a form that can be effectively carried out by an information-processing agent" (https://www.cs.cmu.edu/~CompThink/ (accessed 25/04/18)).

The change in Wings definition reflects the philosophical view which acknowledges the dual ontology of programs. A program is both the text (algorithm) and the object executed by a machine. Wings turn to the thinking agent is correct in that it is focused on the thought processes involved in CT. For instance, when talking about CT many computer science researchers refer to abstraction as a computer science technique, while Wing talks of abstraction as a cognitive tool: "Abstractions are the 'mental' tools of computing." [23] page 3721.

Also, in [25, 26] the authors present their interpretation of CT taking principles from cognitive sciences.

In [25] the author describes his framework based on the computational theory of mind which basically claims that computational processing of information, regardless of the underlying device (electronic or biological), can facilitate and generate cognition (page 21). The author describes the type of computation at two levels, on the one hand, as forms of an associative/distributive processing, that the author also identifies as a bottom-up or inductive and top-down or deductive (pages 22-24) processes, and on the other hand as modelling/simulation types of computation at higher levels (page 27). The author describes how this associative/distributive processing can be found in the explanations of how learning take place in mind (page 23) of different disciplines, namely epistemology, psychology, neuroscience, computer science.

Since the processing of information can be carried out by an electronic or biological underlying device (page 21), his interpretation of CT is then that it is the same as thinking and therefore everyone does CT (page 29).

What distinguishes electronic CT consists of " ... thinking caused by certain uses of electronic computing devices by a biological agent (page 30)". That means that if the thinker uses electronic computing devices, then he/she develops "electronic computational thinking skills".

2 Algorithmic thinking

In response to the literature reviewed here, I begin by stating the following: In order to articulate a theory of the concept of CT the words computational and thinking' must both be consider, and more importantly, the link between the two is key in this regard. Articulating a solid theoretical definition of CT is not

a computational problem, or a psychological one. Each science or discipline is able to answer the questions it is fit to problematise. CT is a problem for the didactics of computer science (didactics of computer science[1]). This is the science of, among other things, the mental processes involved when dealing with the problem of teaching computers to solve algorithmic problems; that is, the problem of learning how to program. In order to contribute to elaborate theoretical founded solutions for those problems, I offer a didactical model from Piaget's theory -Genetic Epistemology- to do research on learning to program.

Piaget studied the mental processes involved in algorithmic thinking in depth, and his theory provides psychological explanations substantiated with broad empirical research. I have extended some principles of his theory to explain the mental processes of CT.

2.1 The general law of cognition

Piaget's theory -Genetic Epistemology- explains the construction of knowledge and offers a model that can be used in all domains and at all levels of development. The central point of Piaget's theory has 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* [10]. 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) [8, 9], and second, from a critical analysis of the history of sciences, elaborated by Piaget and García to investigate the origin and development of scientific ideas, concepts and theories [12].

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 [10]. 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 formalisation. This problem was studied in depth by Piaget in his experiments about genetic psychology. From these he formulated a *general law of cognition* [8, 9], governing the relationship between know-how and conceptualisation, 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

$$\text{C} \leftarrow \text{P} \rightarrow \text{C'}$$

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

[1] Also called didactics of informatic or computer science education.

a task. This reaction is associated to pursuing a goal and achieving results, without awareness neither of actions nor of the reasons for success or failure. The arrows represent the internal mechanism of the (algorithmic) thinking process, by which the subject becomes aware of the coordination of his/her actions (C in the diagram) -that is the method or algorithm she/he has employed- the modifications that these impose to objects, as well as of their intrinsic properties (C' in the diagram), -that is the data structures-. 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 instrument enabling these processes, which he calls *reflective abstraction* and *constructive generalisation* [8, 11], accounting for the principles and mechanisms involved in algorithmic thinking.

Reflective abstraction is described as a two-fold process: First, 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 of success (or failure).

The two phases of the cognitive instrument of generalisation, described in are: inductive generalisation and constructive generalisation. In the first phase, the individual transfers to new objects what has been previously constructed, without taking into account the transformations of the knowledge required for the conditions of the new situation. Because of constructive generalisation, the individual understands the new conditions giving rise to structures, opening the possibility of studying new elements and integrating the constructions of the previous stages as particular cases [11].

3 Extending the general law of cognition: computational thinking

The construction of knowledge about algorithms and data structures is a process regulated by the general law of cognition. Over the years I have investigated the construction of knowledge by novice learners of algorithms and data structures. My research methodology is based on applying Piaget's general law of cognition to make students solve problems (for instance sorting, counting, searching elements [14–16, 19, 17, 13]) and reflect about the method they employ and the reasons for their success (or failure), as a first step towards the conceptualisation of algorithms and data structures.

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. I developed the extension of Piagets general law of cognition as I identified the need to describe 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 themselves.

As Simon Papert says in [20] page 28, referring to the programming of a turtle automata, *"Programming the turtle starts by making one reflect on how one does oneself what one would like the Turtle to do"*. In other words: "programming an automata that solves a problem, starts by making the student reflect on how he/she does herself what he/she would like the automata to do".

To programming an automata solving a problem, the learners have to establish a causal relationship between the algorithm (he/she acting on objects), and the elements relevant to the execution of the program (the computer acting on states). Not only they have to be able to write the algorithm (the text) and represent it as an automata and/or as pseudo-code, but also they have to be able to understand the conditions that make *the computer* run the program.

The generalisation of Papert's words above can be described as: programming an automata starts by making one reflect on

$$\begin{array}{c} \underbrace{\textit{how one does oneself}} \\ \textit{what one would like the automata to do} \end{array}$$

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 description.

By way of analogy with Piaget's law we describe this relationship in the following diagram

$$\begin{array}{ccccc} & & \underbrace{C & \leftarrow & P & \rightarrow & C'} & & \\ newC & \longleftarrow & newP & \longrightarrow & newC' \end{array}$$

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. The subject reflecting on his/her role as problem solver becomes aware of how to do to make the computer solve the problem.

The diagram describes an extension of the law of cognition to encompass not only algorithmic thinking (first row) but also computational thinking (second row).

Piaget identified that the construction of knowledge of methods (algorithms) and objects (data structures) occurs in the interaction between C, P and C'. Likewise, I 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.

My extension of Piaget's law was not initially developed as a theoretical description of 'computational thinking'. Instead, I formulated the theory in order to accurately account for the specificities of the subject instructing a computer to solve a problem.

In the process of applying Piaget's theory to investigate those specificities, I identified the need to extend Piaget's general law of cognition to address the passage from algorithmic to computational thinking.

As is often the case, the need to extend the theory became visible in practice; during an empirical study [18]. 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.

For the study I sought to formulate questions related to actions, that would trigger a thought process which could redirect students' attention, away from themselves and their actions, towards the events taking place in the computer.

This process is what led to the extension of Piagets general law of cognition.

In order to clarify further how my extension of Piagets general law emerged, I will offer a brief description of the empirical study I have made reference to in this paper. This is not an example of the application or verification of a theory. It is a fundamental element of the emergence and development of the theoretical framework I am introducing in this paper. The need to include both theory and practice to develop a framework is not linked to the need to justify the practical use of theory; it simply reflects the dialectical relationship between them.

3.1 The experience prior to the theory

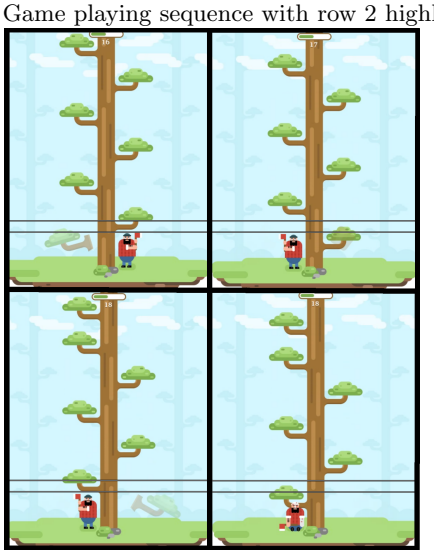
The empirical study which eventually led to the formulation of new theory was carried out in 2017 with average school students between 13 and 15 of an ordinary public High School in Uruguay. The aims of the study were for students; on the one hand, to play a simple video game and to express the rules of *they themselves* playing the game as an algorithm in natural language, on the other hand, to design *an automata* for a program that plays the game, and finally to write and execute *a program* that plays the game. Examples of students answers, some illustrating the success of the new questions, are included.

The game (called Lumber Jack (https://tbot.xyz/lumber)) consists of helping the woodcutter, Jack, to cut a large tree, as shown in figure 1. As Jack hits the tree with the ax, the tree descends a fixed unit. Jack must prevent the branches of the tree from touching his head, if this happens, then the game ends.

The player can move Jack to the left or to the right by pressing two arrow buttons on the screen or the keyboard keys. Each time Jack moves he gives an ax blow on the side of the tree where he has been positioned himself. The player must choose where to position Jack to avoid being hit by the branches of the tree as it descends. It is always possible to dodge the branches that appear since the combination of having branches on the left and on the right is never given.

The students are asked to play the game for a while and then to describe how they play in natural language in their own words. They described actions and objects related to themselves; as exemplified by the quotes below:

Fig. 1. Game playing sequence with row 2 highlighted



1. *I try to play on the phone ... it is uncomfortable ... I use a notebook as support to improve my posture. I get frustrated when I do not succeed and I start again; paying attention to any mistakes in order to correct them. When I start I look up the tree to anticipate movement ... I change the position of my fingers; with index fingers it's better.*
2. *I go slowly when I see a branch and I go faster if there is no branch. I try to prevent. I go slower when it approaches.*
3. *Jack is cutting the trunk; moving to the right and left depending on where the branches appear. When the branch is on the right side Jack runs to the left and when the branch appears on the left side Jack moves to the right.*

The most notable observation at this stage is that the players did not notice that Jack's movement depends on the actions of the player (for instance, at the third quote the player describes Jack's movements as independent from his/her own actions of pressing the keys/buttons.) In other words, there is a lack of awareness of the causal relationship between what the player does and what Jack does.

Keeping in mind my task of inducing students' reflection on how he/she does herself what he/she would like the program to do, I set out to design questions aimed to direct students' attention away from newP (e.g. what they do with their fingers or how they feel) to newC and newC' (Jack's positions, branches states at the row above Jack and what has to be done for not losing). By this I mean, to be aware of the causal relationship between their own actions (perceive the branches, press the keys/buttons) and the events in the computer (Jack's positions, the descending branches, the key/buttons events).

Instead of asking "describe how do you play", then, question 1 **(Q1)** helps direct the players' attention to Jack's different positions on the sides of the tree (newC'). Having done that, questions 2 and 3 now induces students to explicitly write down the rules of the game as inferences (newC).

Q1 Before starting to play, what are the possible positions that Jack can be in, in relation to the tree?
Q2 How do you decide which buttons to press?
Q3 Can you summarise below when success and failure occur, in your own words?

In some excerpts of students' answers to questions 2 and 3 **(Q2 and Q3)** (see below those of level 1) explicit inferences similar to "if Jack was on such a side and the branch on that side, then ... " or "if the pressed button was such and there was a branch, then Jack ... " appear. These reveal awareness of the centres newC and newC'. In contrast, those of level 2, reveal that students' thought remains at newP (that is, focused in his/her-self).

Examples of answers of level 1

- Student 20: Q3: *You succeed when you press the correct buttons, for example: Jack is on the left side and the branch is almost on top of him. You have to go to the right so that the branch does not hit you and you die.*
- Student 7: Q2: *Depending on the position of the branches, the key we are going to press is: branch on the right, we press the arrow on the left so that Jack moves to the left.*
- Student 19: Q2: *You can use left or right buttons; the one you use will depend on where Jack is.* Q3: *Success occurs when, for example, Jack is on the left, there is no branch and we press the left button, or when Jack is on the right, there is a branch and we press the left button.*
- Student 18: Q3: *Success occurs when we press the right button: if Jack is on the left side and there is a branch, you press the other button.*

Examples of answers of level 2

- Student 4: Q2: *I decide to press the keyboard which is easier.*
- Student 3: Q2: *I decide to go to the opposite side in order to cut the tree.*
- Student 5: Q2: *With the mouse you press the arrow to the left or to the right, and pressing the keys on the right which are > and <, or if you use the cell phone; the fingers.*

Central to the task of answering the questions is that the first question is formulated directing students' attention away from they themselves towards the elements in the *world model of the game* (newC and newC'). They were then able to express the causal relationship between their own actions (algorithmic thinking) and the events in the computer (computational thinking). This fact is a direct consequence of applying my framework of the extended general law of cognition in designing the questions.

4 Conclusions

Computational thinking is a term that is broadly discussed and used, but rarely defined. As such, computational thinking has remained diffuse and undistinguished from other related terms; such as, algorithmic thinking. It is for this reason that my study offers a significant novelty in this area. Indeed, with my results I introduce a clear definition of the notion of computational thinking (represented by the second line of the diagram on page 7). Further, this new definition is adequately located in relation to the notion of algorithmic thinking (represented by the first line of the diagram on page 7). My premise is that the depth of my contribution to clarify ideas of computational thinking in educational settings is not clearly visible until it is located within my theoretical framework. Taking principles of Jean Piaget's theory, Genetic Epistemology, I argue that the point of departure for teaching formal knowledge must always be at the level of knowledge that the student has already constructed. In other words; any learning process is built stepwise and is governed by the general law of cognition. In the specific case of learning to program, the process is governed by the new law of cognition as I have formulated it on page 7. Learning how to think computationally, thus, is built stepwise; from the level of actions, to the construction of concepts of computer science (i.e. algorithms and data structures), and finally the construction of formal programs. The theoretical contribution described in this paper provides teachers with a clear description of the term CT, that they can use to help the students learning to program, in a way that respects the process of learning. As a consequence the students are educated to think algorithmically and computationally.

5 Acknowledgements

I am very grateful for comments, suggestions and corrections from to Manuela Cabezas when writing this paper in English. The comments of the anonymous referees are gratefully acknowledged.

References

1. Aho, A.V.: Computation and Computational Thinking. The Computer Journal **55** (2012)
2. Ambrosio, A.P., da Silva, L., Macedo, J., Franco, A.: Exploring Core Cognitive Skills of Computational Thinking. Proceedings of the 25th Annual Psychology of Programming Interest Group Workshop (2014)
3. Borges, K., de Menezes, C., da Cruz, L.: The Use of Computational Thinking in Digital Fabrication Projects - a case study from the cognitive perspective. IEEE: Frontiers in Education Conference (FIE) (2017)
4. Denning, P.J.: Remaining Trouble Spots with Computational Thinking. Communications of the ACM **60** (2017)
5. Grover, S., Pea, R., Cooper, S.: Factors Influencing Computer Science Learning in Middle School. SIGCSE'16 (2016)

6. Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., Settle, A.: Computational Thinking in K-9 Education. Proceedings of the Working Group Reports of the 2014 on Innovation and Technology in Computer Science Education Conference pp. 1–29 (2014)
7. Paulson, L.C.: Computational Thinking is not Necessarily Computational. Communications of the ACM **60** (2017)
8. Piaget, J.: La Prise de Conscience. Presses Universitaires de France (1964)
9. Piaget, J.: Success and Understanding. Harvard University Press (1974)
10. Piaget, J.: Genetic Epistemology, a series of lectures delivered by Piaget at Columbia University, translated by Eleanor Duckworth. Columbia University Press (1977)
11. Piaget, J.: Recherches sur la Généralisation. Presses Universitaires de France (1978)
12. Piaget, J., Garcia, R.: Psychogenesis and the History of Sciences. Columbia University Press, New York (1980)
13. Sylvia da Rosa, B.R.: Didactical ideas in computer science. ITiCSE '16: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (2016)
14. da Rosa, S.: Designing Algorithms in High School Mathematics. Lecture Notes in Computer Science, vol. 3294, Springer-Verlag (2004)
15. da Rosa, S.: The Learning of Recursive Algorithms from a Psychogenetic Perspective. Proceedings of the 19th Annual Psychology of Programming Interest Group Workshop, Joensuu, Finland pp. 201–215 (2007)
16. da Rosa, S.: The Construction of the Concept of Binary Search Algorithm. Proceedings of the 22th Annual Psychology of Programming Interest Group Workshop, Madrid, Spain pp. 100–111 (2010)
17. da Rosa, S.: The construction of knowledge of basic algorithms and data structures by novice learners. Proceedings of the 26th Annual Psychology of Programming Interest Group Workshop, Bournemouth, UK (2015)
18. da Rosa, S.: Students teach a computer how to play a game. LNCS of The 11th International Conference on Informatics in Schools ISSEP 2018 (2018)
19. da Rosa, S., Chmiel, A.: A Study about Students' Knowledge of Inductive Structures. Proceedings of the 24th Annual Psychology of Programming Interest Group Workshop, London, UK (2012)
20. Simon Papert: Papert, S. Mindstorms: Children, Computers, and Powerful Ideas. Basic Books (1980)
21. Tiensuu, A.: Computational Thinking in Regard to Thinking and Problem-Solving. https://tampub.uta.fi/bitstream/handle/10024/83702/gradu06014.pdf (2012), accessed: 2018-04-23
22. Wing, J.: Computational thinking. CACM **49**, 33–34 (2006)
23. Wing, J.: Computational thinking and thinking about computing. Philosophical transitions of the Royal Society **Phil. Trans. R. Soc. A 366,**, 37173725 (2008)
24. Wing, J.: Computational thinking and thinking what and why? https://www.cs.cmu.edu/ CompThink/resources/TheLinkWing.pdf (2010), accessed: 2018-04-18
25. Yasar, O.: Epistemological, Psychological, Neurosciences, and Cognitive Essence of Computational Thinking. Journal of Research in STEM Education pp. 19–38 (2016)
26. Yasar, O., Maleikal, J., Veronesi, P., Little, L.J.: The essence of computational thinking and tools to promote it. American Society for Engineering Education (2017)