International Journal of Education and Development using Information and Communication Technology (IJEDICT), 2017, Vol. 13, Issue 2, pp. 166-184

Computational algorithmization: Limitations in problem solving skills in computational sciences majors at University of Oriente

Antonio S. Castillo
General Teaching Hospital "Dr. Juan Bruno Zayas Alfonso", Cuba

Isabel A. Berenguer, Alexander G. Sánchez, and Tomás R. R. Álvarez
University of Oriente, Cuba

ABSTRACT

This paper analyzes the results of a diagnostic study carried out with second year students of the computational sciences majors at University of Oriente, Cuba, to determine the limitations that they present in computational algorithmization. An exploratory research was developed using quantitative and qualitative methods. The results allowed verifying insufficiencies that have their base in the processes of interpretation and understanding of the problematic situations and in the insufficient design of algorithms, using pseudocodes, before implementing the solutions in a programming language.

Keywords: *Computer programming; computational algorithmization; problem solving.*

INTRODUCTION

The process of computerization of society has gained a great boom in recent times by encouraging the application of Information and Communication Technologies (ICT) to different spheres and sectors of society, in order to achieve greater efficiency through the optimization of resources and the increase of productivity in these areas (Salgado, Alonso & Gorina 2014). For developing countries, such as Cuba, this purpose is a challenge that has led them to identify the need to introduce ICT into the social practice and to achieve an informatic culture that facilitates sustainable development. However, to carry out this purpose requires competent professionals, capable of acquiring that culture and developing it from the ways of acting in their profession (Fergusson et al. 2015).

This requirement has resulted in Cuba have included computer courses in the curriculum of numerous university major, such as bachelor degrees in Mathematics, Physics, Biology, among others, and Mechanical, Biomedical and Electrical engineering. They have also created careers whose object of study is more specific and related to the informatic culture, such as, Computer Science, Computer Engineering, Information Systems, Information Technology and Software Engineering, called computational sciences (ACM 2005).

Consequently, it is hoped that the professionals who graduate from all these careers have appropriated the main scientific and technical advances related to informatic. In addition, in the case of the computationals sciences majors, they must have developed skills that allow them to design, write, debug and maintain the source code of computer programs; code that must be written in a specific language and often requires knowledge of various disciplines, specialized algorithms and formal logic; from which these professionals can create programs that exhibit the desired behavior (Salgado, Gorina & Alonso 2013). A valid strategy in this direction is to begin to teach programming using the algorithms as schematic resources through pseudocode as the main language, to represent the model of the resolution of a problem (Arellano et al. 2012; Blanco, Salgado & Alonso 2016).

Considering all of the above, this article aims to determine the shortcomings that are manifested in the teaching-learning process of the resolving computational programming problems and its dynamics of algorithmization in the majors of computational sciences of the University of Oriente, Cuba: Bachelor´s degree in Computing Science, Engineering in Telecommunications and Electronics, Informatics Engineering and Automatic Engineering.

LITERATURE REVIEW

Teaching-learning process of the resolving computational programming problems

The teaching - learning process of resolving computational programming problems has been approached by numerous researchers, who have obtained important results in the search for a way to teach to program so that the student is able to create his own strategies consciously. Thus, authors such as Arellano et al. (2014) and Ma et al. (2011) assert that the fragility of knowledge in programming is due to the lack of a computer mental model that serves as the basis for creating viable algorithms. But although the conception of this model could be considered an alternative to favor the activity of programming, this is not enough to create efficient programs, since other mathematical, logical and computational knowledge must be incorporated, allowing an integrative perspective to address it.

On the other hand, the venezuelan researchers Torres and Torres (2016), have determined that when students solve problems of programming, they manifest lack of abstraction, few mathematical knowledge and deficiencies in the process of reading and interpretation of problems. In addition to this the student is generally focused on the language tool to be used, rather than in the domain of computational logic. Similarly, the mexican researchers Sánchez, Urías and Gutiérrez (2015), affirm that an important factor that hinders the learning of programming is the demands of language syntax, because many times students after they have understood the concepts, try to execute the programs, and by a syntactic error does not work, which causes them frustration and causes them to become discouraged. These authors suggest teaching programming using algorithms in a previous course and then move on to using language.

In the same way, researchers Adu-Manu, Kingsley and Owusu (2013) refer that in programming courses students usually present problems to understand and interpret the problems they are trying to solve, specifying that, in general the learning of programming language is prioritized to the detriment of algorithmic design skills.

Meanwhile, the French researchers Guibert, Guittet and Girard (2006) found difficulties in the conception of computational programs, concluding that learning of the programming cannot be reduced to learn the syntax of a language, but the student must appropriate a correct algorithmic model of how the programs are developed and executed. These authors, rightly, recognize that the didactics of programming must be based on the processes of analysis, interpretation and algorithmic conception.

Similarly, Whitfield et al. (2007) of the School of Computer Science, Liverpool Hope University argue that the resolution of a computational problem requires skills such as the identification of subproblems, recognition of relationships, situations and models that allow development an algorithm for the solution and the translation of the same to the executable code. With that which refers to the need to enhance the process of computational algorithmization, this position is assumed in the present investigation.

However, all these shortcomings in the resolution of programming problems have also been observed in the computational sciences majors of cuban universities, identified in the study by Salgado et al. (2013). Who made a diagnosis between the courses 2003-2004 to 2011-2012 taking as reference two majors: bachelor´s degree in Computing Science and Informatics Engineering, both from the University of Oriente, Cuba. This study revealed the following shortcomings:

- Limitations in the comprehension of the problematic situations that are posed to them and in their respective modelling from the programming.
- Selection and inappropriate use of computational structures that do not allow the verification and validation of the algorithms that are conceived and implemented.
- Inaccuracies in the computational solutions that are given to problematic situations, which do not always satisfy the original requirements.
- Scarce skills in coding computational procedures in a variety of computer languages.

All of the aforementioned investigations, in one way or another, recognize that the teaching-learning process of resolving computational programming problems, has at its center of its difficulties the algorithmization. Taking into account these shortcomings, those that coincide with the observations made by the authors of this paper in the computational sciences majors of the University of Oriente, Cuba, is that the present investigation is developed.

METHODOLOGY

This exploratory study, on the current state of the teaching-learning process of resolving computational programming problems and its dynamics of algorithmization, was essentially based on two diagnostic tools: a survey to second year students of computational sciences majors at University of Oriente and an interview with teachers who teach Programming for these majors. It should be noted, that the results of the survey were expected to be favorable with respect to the basic skills that should be present in the resolution of computational programming problems, since the selected population was made up of students who had already taken the subject of Programming.

In order to determine the type of sampling to be used in the survey, the homogeneity among the programs of the Programming subject for the four computational sciences majors of the above mentioned university was taken into account. More than 80% of the contents coincide in each program (Salgado 2015). Consequently, each major was considered as a conglomerate containing the characteristics of the variables to be studied, which allowed the use of a probability sampling by two-stage clusters (Cochran 1980). However, if one wishes to have a certain level of completeness of the information concerning the behavior of the process under study, it would be very risky to consider only the results provided by the survey to students. Hence it was considered convenient to carry out an interview to teachers who teach classes in computational sciences majors.

Population and sample for the students' survey

For the survey, the population consisted of 182 second year students of the four computational sciences majors at University of Oriente, at the beginning of the 2012-2013 academic year. The composition of the surveyed students was: 31 students of Informatics Engineering, 77 of Engineering in Telecommunications and Electronics, 24 of the Bachelor´s degree in Computing

SURVEY TO SECOND YEAR STUDENTS OF COMPUTATIONAL SCIENCES MAJORS AT UNIVERSITY OF ORIENTE, CUBA

Dear student:

The present survey is part of a study that aims to improve the teaching of the subject Programming. We ask that you please read carefully the information requested and answer all questions truthfully, employing the following categories:

1	2	3	4	5
Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree

Beforehand we extend our appreciation for your valuable contribution. Thank you so much.

Questionnaire

No	AFFIRMATIONS	1	2	3	4	5
1	I read the problems again in my own words.					
2	I do not check the solution to each problem.					
3	I read the problem several times before trying to solve it.					
4	I do not use graphics, tables, equations and functions to represent the problem I am solving.					
5	I distinguish the relevant information from the irrelevant one in each problem before solving it.					
6	I do not begin to solve the problem until I am sure that Ihave interpreted clearly and precisely each of its elements and have an integrated vision of them.					
7	I explore different computational structures (for, if, then, while, among others) before designing a solution algorithm for the problem.					
8	To solve a problem, I design the program without taking into account a logical order to concatenate the computational structures (for, if, then, while, among others).					
9	I estimate the final answer after conceiving a way to reach the solution of the problem.					
10	After finding the solution to the problem, I do not prove other data to execute (run) the program.					
11	After solving a problem, I reflect about the method or methods I used.					
12	I do not design an algorithm before deploying it in a programming language.					
13	I use pseudocode to design the algorithm before implementing it in a programming language.					
14	When solving a problem, start to implement it in a programming language without using pseudocodes.					
15	Dominating the syntax or rules of a programming language is more important than knowing algorithms design.					
16	If a program complies with the syntax of a programming language and runs properly, then it fulfills the purpose for which it was implemented.					

***Figure 1.** Survey to second year students of computational sciences majors at University of Oriente. Source: Created by the authors.*

Science and 50 of Automatic Engineering. However, taking into account the need to economize resources in research, no information was extracted directly from each experimental unit, but it was necessary to select a random sample of the population under study.

In the second stage, a probabilistic sample was chosen, based on a simple random sampling, in each of the selected conglomerates, being made up of 18 undergraduate students in Computing Science Bachelor (75% of the 24) and 23 of Informatics Engineering (74.19% of the 31), for a total of 41 students to be surveyed.

The aim of the survey to the 41 students was to explore the teaching-learning process of resolving computational programming problems and their dynamics of algorithmization, using the operational variable «learning algorithmization for resolving computational programming problems», which is defined as the appropriation and application of algorithmization contents to the efficient and effective resolution of computational programming problems. This variable was operationalized on the basis of the 16 indicators or items that made up the questionnaire applied to students and shown in figure 1.

For the elaboration of the 16 items of the survey, affirmations and negations were used, in order to reduce bias in the opinion of students, which is introduced by using a single type of item (Bayona et al. 2005). Then, in order to unify the answers, it was worked on the transformation of the negations presented in the even items (I-2, I-4, I-6, I-8, I-10, I-12, I-14 and I-16). So, all resulting items were elaborated in an affirmative way to facilitate the data analysis of the final information (see figure 1).

In order to calculate the confidence of the survey, the split-halves method was used (Hernández, Fernández & Batista 2014), obtaining a correlation coefficient equal to 0.78. Therefore, the confidence of the instrument is considered acceptable.

Information processing of the students' survey

The answers are classified according to each of the 16 items previously declared, considering the Likert scale (ordinal), with five levels of response, shown in figure 1. In addition, for the information processing of the survey, four classes were structured considering the mean and the classification thresholds (Gorina & Alonso 2012) those shown below:

Class A: $\overline{C_j} \in [1, 2)$ Very unfavorable zone.

Class B: $\overline{C_j} \in [2, 3)$ Unfavorable zone.

Class C: $\overline{C_j} \in [3, 4)$ Favorable zone.

Class D: $\overline{C_j} \in [4, 5]$ Very favorable zone.

Subsequently, the following estimates were used:

- Overall criterion for a given item, which is expressed from the arithmetic mean of the scores of item j (mean of column j):

$$\overline{C_j} = \frac{\sum_{i=1}^{mj} C_{ij}}{mj}$$

- Degree of student agreement for a given item, expressed as variance (σ_j^2), the standard deviation (σ_j) or coefficient of variation (v_j) of the scores obtained in the item j:

$$\sigma_j^2 = \frac{\sum_{i=1}^{mj}(C_{ij} - \overline{C_j})^2}{mj - 1} \qquad \sigma j = \sqrt{\sigma^2 j} \qquad v_j = \frac{\sigma_j}{\overline{C_j}}$$

It should be noted, that more attention was paid to the coefficient of variation (which is the standard deviation expressed as one percent of the average) thus providing a measure of the magnitude of the variation relative to the size of the quantity being measured. Therefore, the higher the value of v_j, the lower the degree of agreement of the *m* students related to item *j*. It was assumed as plausible value for the cut-off point or threshold of v_j, to accept an adequate agreement of the students in item j, to value 0.25; which represents a quarter of v_j (Martínez & Martínez 2008).

Population and sample for the teachers' interview

The interview was designed in an unstructured and individual way and an interview guide was used as instrument. This interview was conducted in person and recorded. In general, the duration was 30 to 40 minutes. Twenty teachers were chosen who constitute 80% of those who teach Programming in the four computational sciences majors at the Orient University. This selection was based on: years of experience, teaching category and scientific degree. The interview aimed to explore in the teaching-learning process of resolving computational programming problems and their dynamics of algorithmization, for which the following three indicators were taken into consideration:

- Moment of the programming process in which the greatest difficulties are present:
  - Analysis and interpretation of the problem situation.
  - Computational program elaboration.
  - Execution and validation of the program.
  - Interpretation of program results.
- Other aspects considered relevant for the teaching-learning process of resolving computational programming problems.
- How you would like to be taught Programming?

Information processing of the teachers' interview

Upon completion of the field work (transcriptions of the audio recordings and notes taken during interviews) the content of the textual data was analyzed. The information collected was classified according to the aspects foreseen in the guide and simple counts of similar responses, generated with respect to the questions raised in the interviews, were made. Thus, a script was obtained that allowed to develop analytical categories and theoretical explanations.

This structure was applied systematically to all transcripts, to allow encoding the interview data according to their categories, so that the reorganized data could be analyzed. Simultaneously, key comments were selected, which could later be used as textual quotations.

The theoretical methods of analysis-synthesis and induction-deduction have been used in the analytical process so far, and have been able to propose, order, describe and interpret the data by means of concepts and reasoning, assessing and arriving at conclusions that will reveal the main shortcomings that generated the research problem and its possible causes, some expressed by the respondents and others obtained as a result of the interpretation and construction of the researchers' knowledge.

PRESENTATION OF THE RESULTS

Student survey results

A descriptive synthesis of the results obtained for the two sampled clusters (Bachelor's Degree in Computing Science and Informatics Engineering) is presented in Tables 1 and 2 respectively, which show that the results obtained, in general, are favorable, as we expected from these students, who had passed the subject of Programming. However, a more detailed analysis of the data shows that there are difficulties related to items (I-6, I-12, I-13, I-14, I-15 and I-16) that respond directly to algorithmization in the teaching-learning process of resolving computational programming problems.

Tables 1 and 2 present the data obtained for each of the 16 item after the survey to the 18 students of the Bachelor's Degree in Computing Science and the 23 of Informatics Engineering. In addition, it is show the statistical measures that were considered to characterize the generalized criterion and the level of concordance for each indicator of the survey.

Table 1: *Students' ratings of the Bachelor's Degree in Computing Science for each item of the survey. Arithmetic mean of the students' criteria and statistical dispersion measures to evaluate the level of concordance by item. [Source: Created by the authors].*

Item / Stu	I-1	I-2	I-3	I-4	I-5	I-6	I-7	I-8	I-9	I-10	I-11	I-12	I-13	I-14	I-15	I-16
S-1	4	5	5	5	5	1	5	5	4	5	5	5	4	4	5	4
S-2	4	4	3	3	4	2	4	3	4	4	4	2	4	4	2	2
S-3	4	4	5	4	4	1	3	4	4	5	4	1	4	3	4	2
S-4	5	4	5	3	4	1	4	3	4	5	5	3	3	2	3	1
S-5	3	3	5	4	3	2	4	4	4	5	3	2	4	1	3	2
S-6	4	4	5	2	1	1	5	3	3	5	1	2	4	2	3	1
S-7	4	4	5	5	4	2	4	5	4	4	5	5	1	4	4	2
S-8	1	1	3	3	4	4	2	5	3	5	2	2	3	2	3	3
S-9	2	4	4	5	5	2	5	3	2	5	5	3	2	1	3	3
S-10	1	2	1	3	2	4	2	1	1	3	3	2	2	2	4	3
S-11	3	3	5	3	4	2	5	4	5	4	4	4	1	4	3	2
S-12	4	2	4	4	5	4	2	4	2	2	4	1	2	1	4	5
S-13	4	2	4	4	4	2	4	4	4	4	4	4	3	3	3	2
S-14	4	5	5	4	3	1	3	3	5	5	4	2	3	2	4	4
S-15	4	4	4	4	3	2	4	5	5	5	4	4	4	3	3	3
S-16	3	5	5	3	4	1	3	5	5	5	3	1	2	4	3	1
S-17	4	4	5	4	4	1	3	4	4	5	4	5	4	3	4	2

Item / Stu	I-1	I-2	I-3	I-4	I-5	I-6	I-7	I-8	I-9	I-10	I-11	I-12	I-13	I-14	I-15	I-16
S-18	4	4	5	5	4	2	4	5	4	4	5	5	5	4	4	2
$\overline{C}_j$	3.44	3.56	4.33	3.78	3.72	1.94	3.67	3.89	3.72	4.44	3.83	2,94	3.06	2.72	3.44	2.44
σ_j^2	1.26	1.32	1.18	0.77	1.04	1.11	1.16	1.06	1.27	0.73	1.21	2,17	1.35	1.27	0.50	1.20
σ_j	1.12	1.15	1.08	0.88	1.02	1.06	1.08	1.03	1.13	0.86	1.10	1,47	1.16	1.13	0.70	1.10
v_j	0.33	0.32	0.25	0.23	0.27	0.54	0.29	0.26	0.30	0.19	0.29	0,50	0.38	0.41	0.20	0.45
P (+)	12	12	15	11	13	3	12	11	13	16	13	7	8	6	8	3
P (-)	3	4	1	1	2	15	1	3	3	1	2	9	6	8	1	11
P(+/-)	3	2	2	6	3	0	5	4	2	1	3	2	4	4	9	4
MaxP.	5	5	5	5	5	4	5	5	5	5	5	5	5	4	5	5
MinP.	1	1	1	2	1	1	1	2	1	2	1	1	1	1	2	1

Where:
P (+): the number of evaluations performed with scores 4 or 5. **P (-)**: the number of evaluations performed with scores 1 or 2. **P (+/-)**: the number of evaluations performed with score 3. **MaxP** represents the maximum value of the scores and **MinP** represents the minimum value of the scores.

Table 2: *Students' ratings of Informatics Engineering for each indicator of the survey. Arithmetic mean of the students' criteria of and statistical dispersion measures to evaluate the level of concordance by indicator. [Source: Created by the authors].*

Item / Stu	I-1	I-2	I-3	I-4	I-5	I-6	I-7	I-8	I-9	I-10	I-11	I-12	I-13	I-14	I-15	I-16
S-1	4	5	5	1	4	2	5	5	5	3	3	3	1	5	3	2
S-2	2	2	4	4	4	2	4	4	2	4	4	2	2	4	4	2
S-3	4	5	4	4	3	1	3	4	2	5	4	4	4	3	4	3
S-4	4	5	4	3	5	2	3	4	4	5	3	1	4	2	4	5
S-5	5	4	4	4	5	2	4	4	4	5	4	4	4	4	2	2
S-6	5	3	5	4	4	2	2	3	3	5	5	3	3	4	4	2
S-7	3	4	5	5	3	3	2	4	3	4	4	1	4	5	3	3
S-8	4	4	5	5	4	2	4	4	4	3	4	2	2	1	2	1
S-9	4	4	4	3	3	3	4	3	3	5	5	3	2	1	5	5
S-10	4	4	5	5	4	2	4	4	4	4	4	2	2	4	4	3
S-11	5	5	5	2	4	1	5	5	2	5	5	5	5	1	1	1
S-12	4	4	4	3	4	3	2	4	1	5	3	5	4	2	5	5
S-13	5	5	5	4	4	4	2	5	4	2	4	1	2	3	2	1
S-14	4	5	5	4	4	3	4	3	3	5	5	3	3	2	4	2
S-15	5	4	5	3	4	2	4	3	3	4	4	2	3	3	3	2
S-16	5	5	4	4	4	2	5	5	5	3	1	1	1	1	5	1
S-17	4	4	4	3	3	3	3	3	4	5	3	4	4	4	3	2
S-18	2	4	4	3	4	1	5	5	4	5	5	4	3	4	4	2
S-19	4	4	4	2	5	2	4	5	4	5	3	1	3	1	3	5

Item / Stu	I-1	I-2	I-3	I-4	I-5	I-6	I-7	I-8	I-9	I-10	I-11	I-12	I-13	I-14	I-15	I-16
S-20	5	5	5	5	5	1	4	4	4	4	4	3	4	3	2	3
S-21	3	3	5	3	5	2	5	4	3	3	4	2	4	2	4	2
S-22	3	4	5	4	3	2	5	5	4	5	2	2	1	2	5	2
S-23	4	4	4	3	4	1	4	5	3	5	5	4	4	3	3	1
$\overline{C_j}$	4,00	4,17	4,52	3,52	4,00	2,09	3,78	4,13	3,39	4,30	3,83	2,70	3,00	2,78	3,43	2,48
σ_j^2	0,70	0,60	0,26	1,08	0,45	0,63	0,57	1,09	0,98	0,86	1,06	1,68	1,36	1,72	1,26	1,81
σ_j	0,83	0,78	0,51	1,04	0,67	0,79	0,76	1,04	0,99	0,93	1,03	1,29	1,17	1,31	1,12	1,34
v_j	0,21	0,19	0,11	0,29	0,17	0,38	0,20	0,25	0,29	0,22	0,27	0,48	0,39	0,47	0,33	0,54
P (+)	18	20	23	12	18	1	18	16	12	18	16	7	10	8	12	4
P (-)	2	1	0	3	0	17	0	4	4	1	2	11	8	10	5	15
P(+/-)	3	2	0	8	5	5	5	3	7	4	5	5	5	5	6	4
MaxP.	5	5	5	5	5	4	5	5	5	5	5	5	5	5	5	5
MinP.	2	2	4	1	3	1	3	2	1	2	1	1	1	1	1	1

Where:
P (+): the number of evaluations performed with scores 4 or 5. **P (-)**: the number of evaluations performed with scores 1 or 2. **P (+/-)**: the number of evaluations performed with score 3. **MaxP** represents the maximum value of the scores and **MinP** represents the minimum value of the scores.

Table 3 shows, for each indicator and for operational variable under study, the population-level estimate of the mean and variance. In addition, it presents the absolute error, the coefficient of variation and the 95% confidence interval. The last column of the report provides the assessment made on the operational variable studied and its respective operationalization in 16 indicators.

Table 3: *Inferences to the student population of the four computational sciences majors at University of Oriente, based on a probability sampling two-stage cluster. [Source: Created by the authors].*

Indicators	Mean	Variance	Coefficient of Variation	Error	Lower Limit	Upper Limit	Qualitative Interpretation
I-1	3.827	0.014	0.031	0.234	[3.593	4.061]	Favorable, with very favorable features
I-2	3.976	0.017	0.033	0.260	[3.716	4.237]	Favorable, with very favorable features
I-3	4.522	0.002	0.011	0.100	[4.422	4.621]	Very favorable
I-4	3.701	0.004	0.017	0.129	[3.572	3.830]	Favorable
I-5	3.951	0.004	0.016	0.125	[3.825	4.076]	Favorable, with very favorable features
I-6	2.062	0.001	0.016	0.065	[1.998	2.127]	Unfavorable, with very unfavorable features
I-7	3.801	0.001	0.009	0.072	[3.729	3.873]	Favorable
I-8	4.100	0.003	0.014	0.113	[3.986	4.213]	Very favorable
I-9	3.601	0.006	0.022	0.157	[3.444	3.758]	Favorable
I-10	4.446	0.002	0.011	0.097	[4.349	4.543]	Very favorable
I-11	3.900	0.001	0.008	0.062	[3.838	3.962]	Favorable

Indicators	Mean	Variance	Coefficient of Variation	Error	Lower Limit	Upper Limit	Qualitative Interpretation
I-12	2.856	0.004	0.021	0.119	[2.737	2.975]	Unfavorable
I-13	3.080	0.001	0.009	0.056	[3.024	3.137]	Favorable
I-14	2.807	0.001	0.009	0.048	[2.759	2.855]	Unfavorable
I-15	3.503	0.001	0.008	0.056	[3.447	3.558]	Favorable
I-16	2.509	0.000	0.008	0.040	[2.469	2.549]	Unfavorable
Operational variable	**3.540**	**0.001**	**0.008**	**0.060**	**[3.480**	**3.600]**	**Favorable**

Results of the interview to teachers

For the first indicator below are some of the teachers' answers:

- «(…) It does not matter the programming language in which the algorithm designed at the stage of program elaboration is implemented, because sometimes the C ++ syntax delays students with aspects that are not important (…)» (T-20).

- «(…) the programming language used to implement and execute the program is not important because the algorithm can be adapted to any of the existing ones. (…)» (T-19)

- «(…) in the programming is not relevant the syntactic part that is carried out in the execution, because that is the responsibility of the compilers (...) no matter the programming language but the algorithm developed (...) make the program is mechanical, not the design (…)» (T-15).

- «(…) the students present difficulties to algorithmize, evidenced in that it is difficult for them to find the mathematical solution, since they do not have a good mathematical training and they lack elements of the abstract thinking that allow them to take the task to algorithms to obtain the solution of the problem (…)» (T-1).

- «(…) most do not know how to algorithmize and have problems in mathematical modeling. Students do not check the algorithms manually to verify their solution (they do not make a semantic check) (…)» (T-9).

- «(…) many do not know how to algorithmize, and present serious problems to implement in a language. They do not know how to analyze and interpret a problem (...) they do not design the algorithm correctly (…)» (T-11).

- «(…) emphasis should be placed on the mathematical comprehension and modeling of the problem, (...) it is always necessary to perform a previous low level modeling where the relationships between the objects or variables of the problem to be solved are evidenced and then move on to a formal mathematical modeling, previous to the algorithmization (…)» (T-12).

- «(…) many students do not know how to analyze and interpret a problem (…)» (T-7).

- «(…) most students do not adequately represent the problem situation through mathematical structures. (T-4).

For the second indicator below are some of the teachers' responses:
- «The use of mathematical packages should be enhanced when teaching programming (…)» (T-5)

- «The mathematical program is not the same as the computational one, since in the first one an input is given and a result is obtained, and in the computational program a generalization is sought in such a way that for any data the same result is obtained» (T-7)
- «(…) teaching could be enhanced by the use of some software that shows the operation of the computer» (T-2)
- «(…) it would be very helpful to have educational software for the subject, which would allow the student to actually observe what happens in the machine when an algorithm is executed, this would help him a lot when he had to abstract himself to design an algorithm» (T-6)
- «(…)you cannot teach to solve programming problems with instructions only, but through abstractions, so that the student has a complete idea of the process» (T-7)
- «(…) students should be given at all times notions on how to optimize the algorithm, even if they are very basic, to create a culture of computational optimization» (T-6)

For the third indicator below are some of the teachers' responses:

- «Dedicating more time to algorithmization, for example, a semester where they only design algorithms in pseudocodes and flowcharts (…)» (T-1).
- «Acquiring a better preparation in mathematical knowledge to facilitate the creation of algorithms (…)» (T-3).
- «Ensuring that high school students are taught to reason when solving a problem» (T-4).
- «(…) ensuring that high school students are taught to design algorithms for the computer, although not implemented, with the goal of developing logical thinking (…)» (T-18)
- «(…) increase the hours in computer labs once the student starts programming (…)» (T-4)

DISCUSSION AND ANALYSIS OF THE RESULTS

About the Student Survey

From the results of the survey it was estimated that the average deviation of 3,540 for the population data, is 0.001 units of the scale. The absolute error committed was 0.060, being lower than the maximum permissible error or minimum tolerable precision to the estimator of the population mean used, which was set at e = 0.11.Thus, with a 95% confidence, it can be stated that the estimated population mean is located in the confidence interval [3,480; 3,600]. Therefore, it can be concluded that the inferences to the student population of the computational sciences majors at University of Oriente were **favorable** and the limitations were mainly found in items I-6, I-12, I-14 and I-16, fundamentally related to the computational algorithmization process. The results of the indicated items are described below:

In item I-6 the mean reached is 2.062, which makes it classified as **unfavorable** with latent features of **very unfavorable**. These results indicate that there are difficulties in *understanding and interpreting a problem situation*, since students begin to solve a problem situation without being totally clear of what is required and this often leads to solve a problem different from the one that arises. Likewise, other researchers have obtained similar results to the present study, such as Apiola and Tedre (2012), who used group interviews and quantitative surveys identified that students often lack skills such as comprehension and resolution of problems, which are necessary to learn computer programming.

Also noteworthy are Adu-Manu, Kingsley and Owusu (2013) who conducted student surveys and interviews with teachers with the objective of determining the limitations in teaching and learning methodologies of programming. Concluding that most of the students lack the skills to analyze and solve computational problems and that present problems for the understanding and representation of the concepts in the main themes of the programming.

This difficulty in the understanding and interpretation, although it has been worked didactically from different sciences, requires a particular treatment from the Didactics of Computational Programming, since students must be trained to perform effective analysis, which leads them to understand and interpret a problem situation before of proceed to its mathematical-computational resolution.

The items I-12 and I-14, corresponding to "*designing the algorithm before implementing it in a language*" and "*resolving a computational programming problem directly in a language without using pseudocodes*", obtained values 2.856 and 2.807 of the estimate for the population mean, classified as **unfavorable**. These values reveal that the majority of students do not create an algorithm or use pseudocodes before implementing the solution in a programming language, which accounts for the insufficient valuation they make of the importance of carrying out the design of the algorithm through pseudocodes to have a successful performance in solving computer programming problems.

These results are similar to those reported by Vargas, Pérez and Blanco (2014) of the University of Informatics Sciences, Cuba, who, with the objective of estimating students' skill to develop algorithms, analyzed the semester reports of the Programming subject and surveyed more of 1000 students of the Informatics Sciences Engineering major. By means of these instruments, they determined that students have little skill to develop algorithms of medium complexity, do not dominate a problem solving methodology and are not able to develop a viable model or structure to solve the problem and abstract the different behaviors of a task.

Also, researchers Dillon, Anderson-Herzog and Brown (2014), in a study conducted with students of the University of Alabama, in majors of Computer Science, Electrical Engineering and Computer Engineering, using surveys, determined the advantages of teaching Students to program with tools without syntax, as visual environments could provide students a lower learning curve for the development of algorithmization skills, and may create limitations in their mental representation of programming. This result reinforces what we obtained in the surveys with the students of the computational sciences of the University of Oriente, referring to items I-12 and I-14, which evidences the need to develop in students the skills to program, before using a high-level programming language.

In item I-16, an average of 2.509, classified as **unfavorable**, was obtained, relative to "*when a program complies with the syntax of a language and its execution is correct, it fulfills the objective for which it was implemented*". This reflects that most students focus on the domain of the syntactic part of computational language, to the detriment of the semantic aspects that are inherent in any programming problem, that is, once students verify that the program is executed, they assume that it fulfills the desired intentionality, without performing a detailed semantic validation of the conditions of the initial problem situation.

The researchers Adu-Manu, Kingsley and Owusu (2013) obtained evidence, in the study cited above, that students present many difficulties to learn the syntax of programming languages, which limits them to correctly understand the programming problem they try to solve. These researchers suggest using flowcharts and algorithms to teach concepts and to help students understand the ideas that are transmitted from the teacher to the student. This, according to them, allows the student to understand the problemic situation before coding in a language, since direct encoding in most cases confuses the student.

Likewise, the results obtained in the present study coincide with those found by mexican researchers Sánchez, Urías and Gutiérrez (2015), who conducted interviews with students and professors of the major Computational Systems Engineering from the Indigenous Autonomous University of Mexico, with the aim to determine the problems that are manifested in the learning of programming. These researchers observed that when students edit the code in a development environment, if the compiler does not show the syntax errors, they think the problem is already solved, otherwise they feel they do not know how to program. So they actually lose more time with the syntactic part, than in the design of the program.

For the interpretation of the results, the bipolar character of the scale (positive and negative) has been taken into account, analyzing only the negative traits. However, it is useful to deepen the qualitative levels in which the positive pole is structured, with the intention of discerning those aspects in which the process under study can be further refined.

Consequently, items I-1, I-2, I-5, I-4, I-7, I-9, I-11, I-13 and I-15 were evaluated as **favorable** and in addition the first three reached features of **very favorable**. Although these results are positive, it is considered necessary to continue to improve the processes related to the comprehension of the initial problem situation, the mathematical-computational representation of this situation (in particular the algorithmic representation using pseudocodes) and the final syntactic-semantic validation. This is evidenced by the fact that students do not yet reach a **very favorable** evaluation in items I-4, I-7, I-9, I-11, I-13 and I-15, which accounts for the need to continue perfecting the processes.

However, based on the statistical inference, items I-3, I-8 and I-10 reached **very favorable** assessments, which refer to the "*repeated reading of the problem situation*", to "*the logical order for using computational structures"* and "*testing of the solutions with multiple sets of data*". In general, students pass repeatedly and automatically from the reading of the problem to their computational implementation, without realizing a totalizing comprehension of the initial problem situation, so they really do develop reading, but do not necessarily reach an adequate comprehension. In addition, basic programming structures and their logical order are learned, but frequently, in the face of a specific problem situation, they do not manifest skills for the identification, selection and hierarchization of such structures, since students find it easier to automate the functioning of the structures than apply them to problematic situations. This is explained by the fact that the first skills are clearly reproductive, while the latter are productive, requiring the concurrence of an algorithmic-computational thinking.

Something that should be noted, is that it is significant that these limitations remain in second year students, who have already studied and passed the subject of Programming and still undervalue the potential of the algorithmization to structure the process of resolving computational programming problems. However, this corroborates the fact that there is an abandonment of the algorithmic didactic approach to develop the dynamics of this process, prioritizing those approaches focused on direct work with a high-level programming language.

About the interview to teachers

Summarizing the essential aspects of the interview, one can see that in the indicator "*moment of the programming process in which the greatest difficulties are confronted*", the most relevant responses of teachers can be classified according to two limitations: "the application of a high-level language for programming without conceiving and employing a previous algorithm" and "the abilities to algorithmize".

Regarding the first insufficiency, teachers agree that the use of a high-level programming language to teach directly to program is not appropriate, because the student wastes too much time learning the syntax and does not spend time to create and develop algorithmization skills that allow solving problems correctly. This view of teachers is shared by dissimilar researchers, highlighting Arellano et al. (2014) who, in studies carried out at the National University of San Luis, Argentina, with students of the Electronic Engineering major, corroborated the good results obtained when teaching programming without using high-level languages directly, since using pseudocodes students were able to better understand the logic of algorithms, as a step prior to programming in a more complex integrated programming environment and in formal programming languages such as C ++ or Java. These authors also analyze the use of softwares to learn to algorithmize, but without having to deal with the strict peculiarities of the syntax, observing that they develop in the students greater abilities for the construction of algorithms.

Teachers also consider that, when the algorithm is designed, its translation into a specific language is quite simple and as there are now compilers that point out the syntactic errors, the problem is reduced to that the algorithm fulfills the desired intentionality. On these opinions, authors like Martínez and Fariñas (2012), have obtained similar results, affirming that the contents referring to languages or software for specific uses, come to occupy a second plane, when searching for the solution of a computational problem applying heuristic resources (rules, strategies, principles) and algorithms (known basic procedures); that is, the solution is first modeled by an algorithmic description and then implemented in a programming language.

In the second case, regarding the insufficiencies in the skills to algorithmize, the teachers recognize the algorithmization as an essential part of the programming process, framed in the stage of elaboration of the program, as a generalizing structure, allowing its implementation in any programming language. They also emphasize the previous stage of analysis and interpretation of the problematic situation, evidencing the pertinence of designing an algorithm that reflects the solution of the problem that is trying to solve, before going to its implementation.

The previously mentioned researchers, Sánchez, Urías and Gutiérrez (2015), also concluded in their study, that it is necessary to teach students to algorithmize, before using some language, stating that in this way the student develops the capacity for logical reasoning to resolve problems using computers. Thus, they propose the inclusion of a subject to develop algorithms before to program object-oriented.

On the other hand, the teachers considered as relevant the formation of an abstract thinking in terms of programming and skills to optimize the algorithms. These opinions are shared by others researchers, among them Arellano et al. (2012), who assert that the current trend is to use software tools as didactic support to facilitate teaching-learning of algorithms. They also explain that at university level the use of computational tools based on pseudocode representations or flow diagrams is more frequent, and they propose the softwares: PSeInt, RAPTOR and DFD, which enhance the development of skills in the design of algorithms.

Likewise, our results agree with the studies argentine researchers Arellano et al. (2014), who propose for the teaching of programming to use various softwares to introduce the student to the notion of algorithm, to graphically complement its resolution, to execute it and to debug it, thus favoring the creation of computational thinking. Similarly, Kordakia, Miatidisb and Kapsampelisa (2008) investigated the apparent feasibility of modeling fundamental programming concepts, focusing on value and reference assignment to teach programming, noting that many students appeared to have mental models "non-viability" of these fundamental concepts and that those students who had viable mental models had significantly better performance in programming tasks. This reinforces the opinions of teachers interviewed, related to the need to use computational products to support the teaching-learning process of programming.

Finally, when asked about "*how you would like to be taught Programming*?", the teachers answered in three very important directions, one referred to the increase in work in algorithmic training (whether using pseudocodes, flowcharts or software to algorithmize), another directed towards the reinforcement of mathematical formation (in previous and higher education) and the last one aimed at beginning the teaching of the programming from previous educational levels to the university.

These opinions of teachers regarding the need for algorithmic and mathematical formation have been corroborated in studies carried out by researchers Zamora, Orús and Díaz (2010), in the courses of Bachelor in Computing Science and Bachelor in Mathematics in the University of Oriente, Cuba. These authors, through the implicative statistical analysis, deepened in the learning of the students of both majors, being able to conclude that when a student who starts in the programming succeeds in appropriating the contents of the mathematical analysis and algebra subjects, has a great probability to approve the subject of Programming.

In this same order of ideas, the cuban researchers Urrutia and Álvarez (2009) from the University of Havana, and Vargas, Pérez and Blanco (2014) from the University of Informatics Sciences, both from Cuba, analyzed how the disciplines Mathematical Analysis (in the first case) and Linear Algebra (in the second), can them contribute to basic skills in students who are learning to program, such as modeling, representation and algorithmization using solid mathematical knowledge allow to perform computational programming tasks. On the other hand, the researcher Yasar (2013), concludes from studies carried out, the convenience of mathematically modeling the problem to be solved, because when a mathematical model is already available, the objective is to find a solution in Form of algorithm, for which pseudocodes can be used, forgetting the syntax of formal languages.

However, as regards the teaching of programming from pre-university level to the university level, cuban researchers Martínez, Pereira and González (2014) agree on their importance and affirm that in Cuba since the 12th grade the first Programming concepts are introduce. This is a good practice, the problem is that Visual Basic is used, reason why we consider that the language should be changed, since it has been shown that the results in the programming are not good when the students begin the computational sciences majors in the universities.

In addition, Spanish researchers Basogain, Olabe and Olabe (2015) say that in several countries such as England, Canada and the United States, the teaching of programming has been included in the primary and secondary levels, often through programming environments designed for novice users. This has allowed students to gain greater programming skills before they start studying at universities.

Analytical triangulation of the results of the applied instruments

Once the two instruments were applied, the student survey and the interview to Programming teachers, we proceeded to perform an analytical triangulation with the objective of contrasting and corroborating the insufficiencies in the teaching-learning process of resolving computational programming problems, for which only the indicators evaluated as **unfavorable** were used, as suggested by Gorina and Alonso (2012).

In the case of item I-6, it was possible to appreciate a correspondence between the students' evaluations and the opinions of the teachers, in both cases it is concluded that there are insufficiencies in the moment of comprehending and interpreting a problem situation. This stage of "comprehension" is often underestimated by students, which is why mistakes are common during the process of solving a problem situation. In this direction, many researchers have arrived at similar conclusions based on observations made to the students' performance, among them Adu-Manu, Kingsley and Owusu (2013), Apiola and Tedre (2012), Arellano et al. (2012), Yasar (2013).

In the case of items I-12 and I-14, the evaluations made by the students show an insufficient recognition of the importance of performing the algorithm design using pseudocodes, with the objective of having a successful performance in the resolution of computational programming problems. This result coincides with those obtained by the researchers Arellano et al. (2014), based on surveys conducted to students of the Electronic Engineering major of the National University of San Luis in Argentina, which conclude that there is an abandonment of the algorithmic approach in the teaching of programming.

In contrast to the above, most of the teachers interviewed agree on the importance of recovering the teaching of programming through the algorithmic approach, using pseudocodes or flow diagrams, given that this allows to focus on the student's algorithmic thinking, without wasting too much time learning the syntax of high-level programming languages. This opinion is shared by other researchers such as Pinales and Velázquez (2014), who consider that the students who are just beginning their studies in the area of programming should be offered a series of representative problems, algorithmically solved, using pseudocode or flow diagrams, which helps develop appropriate logical thinking in students. Also, teachers Tutillo and Ferrer (2015), based on studies conducted in technical universities in Ecuador, agree that it is a necessity for students to dominate algorithm techniques using pseudocodes before moving on to implementation in a high-level programming language.

However, in item I-16, students' evaluations point to a greater domain of the syntactic part of the computational language than to the semantic aspects of the algorithmization, which reduces the efficiency in the resolution of computational programming problems. Nevertheless, most teachers point out that the most important thing is to be able to conceive the algorithm, since its translation into a specific language is not complex; but the practices of the teaching-learning process of the computational programming show a very different reality to this view of the teachers, connoting itself the teaching of the programming language of high-level, as opposed to the ways of stimulation for the formation of an algorithmic thinking. This has been reaffirmed in the results of numerous investigations, highlighting those carried out by Arellano et al. (2014), Cetín (2013), Kinnunen and Simon (2012), Martínez and Fariñas (2012), Sánchez, Urías and Gutiérrez (2015).

In summary, what has been analyzed in this paper shows the need to transform the teaching-learning process of resolving computer programming problems, prioritizing algorithmization using pseudocodes or flow diagrams, for which it will be pertinent to elaborate theoretical and methodological proposals that reveal the essential aspects of the cited process.

CONCLUSIONS

The results of the applied diagnosis made possible to point out that the main limitations of the teaching-learning process of resolving computational programming problems are related to the processes of interpretation and comprehension of problem situations and to the recognition of the need to design algorithms in pseudocodes before implementing the solutions of the mentioned situations in a programming language.

The systematization carried out, based on the survey of students of the computer sciences majors and the interview to programming teachers, reveal the need to expose the specific aspects that distinguish the dynamics of algorithmization in the teaching-learning process of the resolution of computational programming problems, through an integrative and coherent logic, that allows to reach higher levels of interpretation of the essence of the problematic situation that is approached.

The process of analytic triangulation revealed an insufficient use of the algorithmic didactic approach to develop the dynamics of the teaching-learning process of resolving computational programming problems, as well as the predominance of approaches that prioritize the teaching of programming from working directly on a language, which accounts for the need to introduce new relationships that enable the formation of computational algorithmization.

REFERENCES

ACM´05. 2005. “ACM / IEEE-CS Computing Curricula 2005. The overview report”. Available from: en: http://www.computer.org/education/cc2005/ironman/cc2005/index.html [8 January 2016].

Adu-Manu, K, Kingsley, J. & Owusu, PY. 2013. “Causes of failure of students in computer programming courses: The teacher-learner Perspective”, *International Journal of Computer Applications*, vol. 77, issue 12, pp.27-32.

Apiola, M. & Tedre, M. 2012. “New perspectives on the pedagogy of programming in a developing country context”, *Computer Science Education*, vol. 22, issue 3, pp.285-313.

Arellano, JJ., Nieva, OS., Solar, R. & Arista, G. 2012. “Software para la enseñanza-aprendizaje de algoritmos estructurados”, *Revista Iberoamericana de Educación en Tecnología y Tecnología en Educación (TE& ET)*, no. 8, pp. 23-33.

Arellano, N., Fernández, J., Rosas, MV. & Zúñiga, M. E. 2014. “Estrategia metodológica de la enseñanza de la programación para la permanencia de los alumnos de primer año de Ingeniería Electrónica”, *Revista Iberoamericana de Educación en Tecnología y Tecnología en Educación (TE& ET)*, no.13, pp. 55-60.

Basogain, X., Olabe, MÁ. & Olabe, JC. 2015*. “Computational thinking trough programming: a learning paradigm”, Revista de Educación a Distancia*, vol. 46, no. 6, pp.1-33.

Bayona, JA., Hurtado, C., Ruiz, IR., Hoyos, A. & Gantiva, CA. 2005. “Attitudes toward the sale and consumption of psychoactive substances into the national university of Colombia”, *Interamerican Journal of Psychology*, vol. 39, no. 1, pp. 159-168.

Blanco, A., Salgado, A. & Alonso, I. 2016. “Abilities for the computational algorithmization in the Licentiate in Education: specialty education labor-informatics”, *Revista Maestro y Sociedad: Revista electrónica para maestros y profesores*, vol. 13, no.1, pp. 16-28.

Cetin, I. 2013. “Visualization: a tool for enhancing students’ concept images of basic object-oriented concepts”, *Computer Science Education*, vol. 23, issue.1, pp. 1-23.

Cochran, WG. 1980. “Muestreo”, Trillas, Mexico.

Dillon, E., Anderson-Herzog, M. & Brown, M. 2014. “Teaching students to program using visual environments: Impetus for a faulty mental model?”, *Journal of Computational Science Education,* vol. 5, issue 1, pp.1-2

Fergusson, EM., Alonso, I., Salgado, A. & Gorina, A. 2015. “Dynamics of the investigative formation process in the career of bachelor's degree in Computing Sciences”, *Revista Didasc@lia: Didáctica y Educación,* vol. 6, no.6, p.89.

Gorina, A. & Alonso, I. 2012. “A system of methodological procedures to improve the processing of information in social researches”, *Revista Didasc@lia: Didáctica y Educación*, vol. 3, no.6, pp. 91-108.

Guibert, N., Guittet, L. & Girard, P. 2006. “Performances et usages d’un environnement d’apprentisage de la programmation basé sur exemple”, *ERGO'IA*, pp. 103-110.

Hernández, R., Fernández, C. & Batista, P. 2014. “Metodología de la investigación”, 6th edn, McGraw-Hill, Mexico.

Kinnunen, P. & Simon, B. 2012. “My program is ok-am I? Computing freshmen's experiences of doing programming assignments”, *Computer Science Education*, vol. 22, issue 1, pp.1-28.

Kordakia, M., Miatidisb, M. & Kapsampelisa, G. 2008. “A computer environment for beginners’ learning of sorting algorithms: Design and pilot evaluation”, *Computers & Education*, vol. 51, issue. 2, pp. 708-723.

Ma, L., Ferguson, J., Roper, M. & Wood, M. 2011. “Investigating and improving the models of programming concepts held by novice programmers”, *Computer Science Education*, vol. 21, issue 1, pp. 57-80.

Martínez, JA. & Martínez, L. 2008. “Determining the most unfavourable variance to calculate the measurement scale imprecision factor, and extension to other types of sampling methods”, *Revista Psicothema*, vol. 20, no. 2, pp. 311-316.

Martínez, R., Pereira, M. & González, R. 2014. “Feasibility of Python in teaching programming”, *Revista cientítfico pedagógica: Mendive*, vol. 12, no.46, pp.179-186.

Martínez, S. & Fariñas, JL. 2012. “The competence elaborating informatics programs in the teaching-learning process of the discipline programming language and techniques”, *Revista Didasc@lia: Didáctica y Educación*, vol. 3. no.2, pp. 125-144.

Pinales, FJ. & Velázquez, CE. 2014. “Algoritmos resueltos con diagramas de flujo y pseudocódigo”, Universidad Autónoma de Aguascalientes, Mexico. Available from: http://www.uaa.mx/direcciones/dgdv/editorial/docs/algoritmos.pdf [1 January 2015].

Salgado, A .2015. “Logic-algorithmic dynamic of the process of resolving computational programming problems”, PhD thesis, University of Oriente, Cuba.

Salgado, A., Gorina, A. & Alonso, I. 2013. “Model of the algorithmic–logic dynamic to solve computer programming problems”, *Revista Educare*, vol. 17, no. 1, pp. 27 – 51.

Salgado, A., Alonso, I., Gorina, A. & Tardo, Y. 2013. “Algorithmic logic to solve computational programming problems: a didactic proposal”, *Revista Didasc@lia: Didáctica y Educación*, vol. 4, no. 1, pp. 57-76.

Salgado, A., Alonso, I. & Gorina, A. 2014. “Exemplification of the solution algorithmic of problems of programming computacional”, *Revista Didasc@lia: Didáctica y Educación*, vol. 5, no. 4, pp. 15-36.

Sánchez, JE., Urías, M. & Gutiérrez, BE. 2015. “Analysis of learning problems of object-oriented programming”, *Ra Ximhai*, vol. 11, no. 4, pp. 289-304

Torres, M. & Torres, M. 2016. “ESENCi: Teaching strategy for computer problem solution using problem based analysis”, *Revista Educare*, vol. 20, no. 2, pp. 78-102.

Tutillo, ID. & Ferrer, M. 2015. “Platform theoretical referential of the process of teaching learning of the programming logic in the formation of technologists in analysis of systems”, *Santiago*, vol.137, pp. 589-605.

Urrutia, I. & Álvarez, V. 2009. “Un acercamiento a las nociones de competencias básicas que la Disciplina Análisis Matemático debe contribuir a desarrollar en los estudiantes de Ciencias de la Computación”, *Boletín de la Sociedad Cubana de Matemática y Computación*, vol.5, número especial 2009.

Vargas, A., Pérez, OL. & Blanco, R. 2014. “Algorithmization skill development: a view from Science, Technology and Society studies”. Proceedings of the first International scientific conference UCIENCIA 2014. University of Informatics Sciences. Cuba. Available from: https://uciencia.uci.cu/sites/default/ files/public/p3499-ponencia-1062_ 0.pdf. [12 June 2014].

Whitfield, AK., Blakeway, S., Herterich, GE & Beaumont. C. 2007. “Programming, disciplines and methods adopted at Liverpool Hope University”, *ITALICS*, vol. 6, issue 4, pp.145-168

Yasar, O. 2013. “Computational Math, Science and Technology (C-MST). Approach to General Education Courses”, *Journal of Computational Science Education,* vol. 4, issue 1, pp. 2-10.

Zamora, L., Orús, P. & Díaz, JR. 2010. “El análisis estadístico implicativo, instrumento común de investigación en una experiencia de cooperación multidisciplinar: Visualizar una expresión de discontinuidad del rendimiento académico en estudiantes universitarios de Matemática y Computación usando análisis estadístico implicativo”, *Quaderni di Ricerca in Didattica (Mathematics)*, no.20, suppl 1, pp.451-475.