UNIVERSIDAD DE ORIENTE

CENTRO DE ESTUDIOS DE EDUCACIÓN SUPERIOR "MANUEL F. GRAN"

DINÁMICA LÓGICO – ALGORÍTMICA DEL PROCESO DE RESOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN COMPUTACIONAL

Tesis presentada en opción al grado científico de Doctor en Ciencias Pedagógicas

Autor: MSc. Antonio Salgado Castillo

Santiago de Cuba, 2015

UNIVERSIDAD DE ORIENTE

CENTRO DE ESTUDIOS DE EDUCACIÓN SUPERIOR "MANUEL F. GRAN"

DINÁMICA LÓGICO – ALGORÍTMICA DEL PROCESO DE RESOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN COMPUTACIONAL

Tesis presentada en opción al grado científico de Doctor en Ciencias Pedagógicas

Autor: MSc. Antonio Salgado Castillo

Tutores: Dra. C. Isabel Alonso Berenguer

Dr. C. Alexander Gorina Sánchez

Santiago de Cuba, 2015

AGRADECIMIENTOS

No creo que este texto sea suficiente para recoger el nombre de todas las personas a las que debo agradecer de una u otra forma por su colaboración en la realización de esta tesis, por ello pido disculpas si involuntariamente omito a alguna.

Agradecimientos especiales:

- A mi tutora Dra. C. Isabel Alonso Berenguer, por su profesionalidad, rigor científico, gran dedicación, paciencia, confianza, hospitalidad, por su apoyo incondicional y sobre todo por lo mucho que aprendo cada día con ella y seguiré aprendiendo.
- A mi tutor Dr. C. Alexander Gorina Sánchez, por su exigencia, sus demostraciones constantes de rigor, seriedad científica, ética, profesionalidad y certeras orientaciones, así como, por estimular desde el primer momento la perseverancia en el proceso investigativo y sobre todo por lo mucho que aprendí con él y espero seguir aprendiendo.
- A mi amiga la Dra. C. Elena Torres Barandela, ejemplo de abnegación y perseverancia en nuestro grupo de investigación, quién todavía no sabe decirme que no y ha estado tan preocupada por mí como mi propia madre.
- Al Dr. C. Arquímedes Montoya Pedrón por ser ejemplo de investigador consagrado, cuyas orientaciones me han encaminado en mi desarrollo

profesional e investigativo. Me considero afortunado de haber podido contar con su valiosa amistad, confianza y ayuda incondicional.

- A la dirección del Hospital General “Dr. Juan Bruno Zayas Alfonso” y en especial a la Vicedirectora de Docencia e Investigación Dra. C. María Eugenia García Céspedes por toda la confianza y el apoyo brindado.
- Al equipo de Mella: Taimé, Aniela, Mailen, Miraides, Juan, Yodanys por su permanente entusiasmo.
- A la MSc. Ekaterine Miriam Fergusson Ramírez, la MSc. Jennifer Martínez Mojicar y a la MSc. Tahimy González Rubio por su ayuda incondicional.
- Al Dr. C. Rolando Pavó Acosta por su oportuno asesoramiento y buenos consejos.
- Al Dr. Cs. Homero C. Fuentes, la Dra. C. Eneida C. Matos, la Dra. C. Dalia Bencomo, la Dra. C. Yaritza Tardo y a todos los profesores del CeeS “Manuel F. Gran” que confiaron en mí, me animaron y me orientaron científicamente.
- A las doctoras Silvia S. Cruz y Rosario León por sus certeras oponencias que contribuyeron al perfeccionamiento de esta investigación.
- A todos los amigos, amigas, vecinos y familiares que no he mencionado, pero que no hace falta, pues saben que les estoy eternamente agradecido.

DEDICATORIA

- A mi mamá, Hilda Elena Castillo Moldes por estar siempre a mi lado, guiándome y cuidándome, además estimulándome y apoyándome en todo momento con la convicción de que podría lograrlo.
- A mi hijo Antonio Salgado Maresma (Tikín), que Dios le dé mucha salud.
- A mi papá Antonio Salgado Suárez, dondequiera que estés, espero estar cumpliendo tu sueño.
- A mi esposa Karen Maresma Nicle, por estar siempre a mi lado y apoyar mi constancia y esfuerzo.
- A mis hermanas Lídice y Maryanis por su constante ayuda y preocupación.
- A mi segunda madre, que es inteligencia pura y que ha sufrido y disfrutado cada momento de esta investigación, pero siempre a mi lado, sin soltarme la mano. ¡Gracias!

SÍNTESIS

La investigación parte del **problema científico**: insuficiente aprendizaje de los contenidos de programación, en relación con la interpretación de situaciones problémicas, lo que limita su aplicación eficiente a la resolución de problemas computacionales; considerándose como **objeto** de investigación, el proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional, y como **campo de acción,** la dinámica del proceso de enseñanza – aprendizaje de la algoritmización computacional. El **objetivo** consistió en la elaboración de un sistema de procedimientos didácticos para la algoritmización computacional, sustentado en un modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional. Los principales **aportes** fueron el citado modelo y el sistema de procedimientos didácticos. Se valoró satisfactoriamente la pertinencia y factibilidad científica de los resultados mediante cuatro talleres de socialización con especialistas, la ejemplificación del sistema de procedimientos didácticos y el desarrollo de un experimento pedagógico. La **novedad científica** consistió en haber fundamentado la relación entre la representación matemática de la situación problémica y su generalización pseudocodificada, como condición imprescindible para el desarrollo de un pensamiento algorítmico computacional; revelándose la doble modelación, matemática y computacional, que debe experimentar una situación problémica en el camino algorítmico que conduce a su solución.

		Ingeniería Informática de la Universidad de Oriente.	
	6	Procesamiento de la encuesta a estudiantes del 2do año de las carreras de Licenciatura en Ciencia de la Computación y de Ingeniería Informática de la Universidad de Oriente.	
	7	Guía para la segunda entrevista a profesores de Programación.	
	8	Procesamiento de los datos obtenidos a partir de la segunda entrevista a profesores de Programación.	
	9	Estructura del sistema de procedimientos didácticos para la algoritmización computacional.	
	10	Evidencias de la realización de los talleres de socialización.	
	11	Pruebas aplicadas a los dos grupos bajo estudio (control y experimental).	

INTRODUCCIÓN

El proceso de informatización de la sociedad ha cobrado gran auge en los últimos tiempos al propiciar la aplicación de las Tecnologías de la Información y las Comunicaciones (TIC) a las diferentes esferas y sectores de la sociedad, en aras de lograr una mayor eficacia y eficiencia, mediante la optimización de recursos y el incremento de la productividad en dichas esferas (Salgado, A., Alonso, I. y Gorina, A., 2014a).

Para los países en desarrollo este propósito constituye un reto que los ha llevado a identificar la necesidad de introducir en la práctica social las TIC y lograr una cultura informática que facilite un desarrollo sostenible. Sin embargo, para llevar a la práctica este propósito se requiere de profesionales competentes, capaces de adquirir dicha cultura y desarrollarla desde los modos de actuación de su profesión (Remedios, M. A., 2006).

Este requerimiento ha traído consigo la inclusión de asignaturas informáticas en el currículo de numerosas carreras universitarias, tales como las licenciaturas en Matemática, Física, Química, Biología, etc., y las ingenierías en Mecánica, Biomédica, Eléctrica, etc.; así como en otras carreras de corte social, humanístico y de la salud como Gestión de la Información (Salgado, A., Gorina, A. y Alonso, I., 2013). También se han creado carreras cuyo objeto de estudio es más específico y relacionado con la cultura informática, como son la Licenciatura en Ciencia de la Computación (LCC) y las ingenierías en Informática, Telecomunicaciones y Electrónica (ITE), Ciencias Informáticas y Automática, denominadas ciencias computacionales (ACM, 2005).

Por consiguiente, se aspira que los profesionales que egresen de todas estas carreras se hayan apropiado de los principales adelantos científico – técnicos relacionados con la informática (Tan, J. y otros, 2014). Además, en el caso de las ciencias técnicas, naturales y exactas, deberán haber desarrollado habilidades que les permitan, diseñar, escribir, depurar y mantener el código fuente de programas computacionales; código que debe ser escrito en un lenguaje específico y con frecuencia requiere de conocimientos de varias disciplinas, del dominio del lenguaje a utilizar, de algoritmos especializados y de la lógica formal; a partir de lo cual dichos profesionales podrán crear programas que exhiban el comportamiento deseado (De Lobos, M. E., 2010).

Sin embargo, en la actualidad se presentan insuficiencias en la apropiación de los contenidos de programación, lo que confirman M. Oviedo y F. G. Ortiz (2002), en un estudio hecho en la Academia de Computación de la PIICSA, México, el que corroboró la carencia de habilidades para programar y el bajo aprovechamiento docente en la asignatura de Programación. Así también los investigadores A. Ferreira y G. Rojo (2005), de la Universidad Nacional de Río Cuarto, en Argentina, aseguran que los alumnos del primer año que cursan la asignatura de Introducción a la Algorítmica y Programación, presentan insuficiencias tales como: escasa destreza para desarrollar algoritmos, ausencia de una metodología de resolución de problemas e inexperiencia en el manejo del lenguaje de programación.

Por su parte los investigadores franceses N. Guibert, L. Guittet y P. Girard (2005b) plantean que los estudiantes que se enfrentan por primera vez a la programación presentan problemas para desarrollar un modelo viable o estructura que les permita resolver problemas o crear una estrategia que los resuma todos.

Los profesores de la Universidad de Southampton, H. C. Davis, C. Hugh y S. White (2011), explican que los resultados de promoción en los cursos de Programación son muy bajos, destacando que los estudiantes no son capaces de usar correctamente conceptos comunes. Una opinión generalizadora de esta situación es la que ha sido emitida por J. Käasboll (2002), quien en sus informes asegura que a nivel mundial, entre el 25 y el 80 % de estudiantes que en su primer año de universidad enfrentan cursos de Programación, han desaprobado o han abandonado los estudios de esta materia.

Teniendo en cuenta estas insuficiencias, que manifiestan coincidencia con lo observado por el investigador en las carreras de ciencias computacionales de la Universidad de Oriente (UO), en Santiago de Cuba, se llevó a cabo un diagnóstico en dichas carreras, es decir, en la Licenciatura en Ciencia de la Computación y las ingenierías en Telecomunicaciones y Electrónica, Informática y Automática. Para ello se utilizaron los siguientes medios: análisis de los resultados evaluativos obtenidos por los estudiantes de las cuatro carreras en la asignatura Programación, en los cursos 2003 – 2004 al 2011 – 2012 (ver anexo 1), así como entrevista al 96 % de los profesores que imparten la citada asignatura en dichas carreras (ver anexos 2 y 3).

Del procesamiento de la información obtenida con el diagnóstico se pudieron develar las siguientes **manifestaciones**:

- Escasas destrezas para desarrollar procesos de abstracción y decodificación de situaciones problémicas, con el objetivo de modelarlas desde la programación.
- Limitaciones en la utilización de estructuras computacionales, lo que afecta el correcto diseño e implementación de los programas.
- Imprecisiones en las soluciones computacionales que se dan a las situaciones problémicas, las cuales no siempre satisfacen las exigencias originales.

A partir de las manifestaciones fácticas reveladas y desde la cultura epistemológica y experiencia del postulante, se define como **problema científico**, el insuficiente aprendizaje de los contenidos de Programación, en relación con la interpretación de situaciones problémicas, lo que limita su aplicación eficiente a la resolución de problemas computacionales. Ello es expresión científica de la **contradicción epistémica inicial** entre la apropiación de contenidos de programación y su aplicación a la solución de problemas computacionales.

Entre las **causas** principales de este problema, obtenidas a través del diagnóstico realizado, se revelan:

- Insuficiente concepción científico – metodológica del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional, que no profundiza suficientemente en las especificidades de su lógica didáctica, ni en las relaciones esenciales que lo caracterizan.
- Limitados enfoques teóricos y didácticos de la resolución de problemas de programación computacional, la que es abordada desde perspectivas reduccionistas y mecánicas, que no trascienden la presentación de un gran número de ejemplos análogos, en un lenguaje de programación específico.
- Predominio de sesgos en los enfoques didácticos que abordan la resolución de problemas de programación, al no privilegiar la concepción del ordenamiento lógico que debe preceder a la implementación computacional.

Las causas precisadas indican la necesidad de perfeccionar el proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional, el cual se asume como **objeto de la investigación**. Este proceso de enseñanza – aprendizaje ha sido abordado por numerosos investigadores, los que han obtenido importantes resultados en la búsqueda de una forma de enseñar a programar de manera que el alumno sea capaz de crear sus propias estrategias de manera consciente. Así, autores como N. Arellano y otros (2014) y L. Ma y otros (2011) aseguran que la fragilidad del conocimiento al programar se debe a la falta de un modelo mental de la computadora que le sirva de base para crear algoritmos viables. Pero si bien la concepción de dicho modelo pudiera considerarse una alternativa para favorecer la actividad de programar; ésta no es suficiente para crear programas eficientes, pues deben incorporarse otros conocimientos matemáticos, lógicos y computacionales, que permitan una perspectiva integradora para abordar el mismo.

Por otro lado, las investigadoras argentinas N. Moroni y P. Señas (2005) afirman que la complejidad de los programas que se desarrollan actualmente exige del uso de técnicas efectivas de programación, por lo que debe ponerse énfasis en el diseño previo, ponderando la utilización de los algoritmos como recursos esquemáticos para plasmar el modelo de la resolución de un problema. Si bien estas autoras reconocen el valor del empleo de algoritmos como recurso previo a la resolución de un problema, su perspectiva de análisis no trasciende lo meramente declarativo, sin ofrecer posibles vías para diseñarlos.

Por su parte los franceses N. Guibert, L. Guittet y P. Girard (2005a), constataron dificultades en la concepción de programas computacionales, concluyendo que el aprendizaje para programar no puede reducirse a aprender la sintaxis de un lenguaje, sino que el estudiante debe apropiarse de un modelo algorítmico correcto de cómo los programas se elaboran y ejecutan. Estos autores, de manera acertada, reconocen que la Didáctica de la Programación no puede sustentarse en el aprendizaje de lenguajes de programación, sino en procesos de análisis, interpretación y abstracción de su lógica y concepción algorítmica.

De igual forma, A. K. Whitfield y otros (2007), de la School of Computer Science Liverpool Hope University plantean que la resolución de un problema computacional requiere de habilidades tales como la identificación

de subproblemas, el reconocimiento de relaciones, situaciones y modelos que permitan desarrollar un algoritmo para la solución y la traducción del mismo al código ejecutable. Con lo que remiten a la necesidad de potenciar el proceso de algoritmización computacional desde una perspectiva didáctica, postura esta que es asumida en la presente investigación.

Como se puede apreciar, todas estas investigaciones reconocen que el proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional tiene como centro de sus dificultades la algoritmización. Sin embargo, la generalidad de ellas se queda sólo a nivel del reconocimiento de dicha problemática, sin llegar a proponer soluciones didácticas orientadas a revelar la lógica del proceso de algoritmización computacional.

En resumen, la sistematización epistemológica realizada ha revelado las limitadas propuestas teóricas y metodológicas existentes, las que aún no logran develar las esencialidades didácticas del proceso de algoritmización computacional, desde un sistema de relaciones que caracterice su lógica dinamizadora en la resolución de problemas de programación, lo que deviene **insuficiencia epistémica** de la presente investigación. Por lo que el **campo de acción** se precisa como la dinámica del proceso de enseñanza – aprendizaje de la algoritmización computacional.

De aquí que resulten necesarios nuevos procederes didácticos en el proceso de algoritmización computacional, lo que remite a profundizar en las particularidades de su dinámica. En tal sentido, las perspectivas de análisis para abordar la Didáctica de la Programación, se fundamentan en dos enfoques, uno orientado a la especificidad de un lenguaje particular (Wang, J., Mendori, T. y Xiong, J., 2014; Ramos de Melo, F. y otros, 2014; Pérez, R., 2009; González, W., Estrada, V. y Martínez, M., 2006; Al-Imamy, S., Alizadeh, J. y Nour, M. A., 2006; Moroni, N. y Señas, P., 2004), y otro que emplea un lenguaje algorítmico a través de pseudocódigos para facilitar su posterior implementación (Arellano, N. y otros, 2014; Arellano, J. J. y otros, 2012; Novara, P., 2012; Kordakia, M., Miatidisb, M. y Kapsampelisa, G., 2008; Faouzia, B. y Mostafa, H., 2007; Guibert, N., Guittet, L. y Girard, P., 2006; Ferreira, A. y Rojo, G., 2005; Martínez, Y., 2005).

Los partidarios del primer enfoque proponen el uso de mapas conceptuales, multimedias, tutoriales y libros electrónicos en lenguajes de programación de alto nivel, que faciliten la representación de los programas implementados, así como el diseño, visualización y prueba de estos. En opinión del postulante, estas herramientas aumentan la comprensión de conceptos de un alto nivel de abstracción, pero enfatizan en el dominio de la sintaxis del lenguaje, en detrimento de las habilidades de algoritmización.

En este enfoque destaca la propuesta de los investigadores árabes S. Al-Imamy, J. Alizadeh y M. A. Nour (2006), los que proponen que la enseñanza de la programación se sustente en el uso de plantillas en lenguaje C, afirmando que los estudiantes logran apropiarse de los conceptos principales del lenguaje en poco tiempo. No obstante es criticable que sólo se potencie el aprendizaje de la sintaxis del lenguaje, desdeñando la formación de habilidades de programación referentes al diseño lógico que deben tener los programas.

En este mismo enfoque se reconoce como válida la propuesta hecha por el investigador mejicano R. Pérez (2009), consistente en un software que usa seis lenguajes para potenciar el desarrollo de programas; así los alumnos empiezan a descubrir la sintaxis de los lenguajes y a programar rápidamente, pues el software indica las correcciones sintácticas. Sin embargo, aunque se reconoce esta contribución, el software puede conducir a una mecanización del pensamiento, al emplearse mayor tiempo en aprender la sintaxis que en el diseño lógico del programa, no logrando potenciar las relaciones existentes entre la algoritmización y su posterior codificación en un lenguaje, al no desarrollar habilidades esenciales como la modelación y la representación.

Por su parte los investigadores brasileños F. Ramos de Melo y otros (2014), proponen un sistema tutorial inteligente basado en redes neuronales artificiales para que los estudiantes puedan organizar y personalizar los contenidos en el estudio de la Programación, lo que a decir de estos autores permite un mejor uso del contenido y el conocimiento. Esta propuesta se dirige a potenciar una estructuración del aprendizaje guiada por el estudiante, lo que no se considera totalmente acertado, pues la práctica ha demostrado que la complejidad de la programación exige de la mediación del profesor para la conducción del uso correcto de las herramientas computacionales que sirven de apoyo al proceso de enseñanza – aprendizaje.

Así mismo, los investigadores cubanos W. González, V. Estrada y M. Martínez (2006) proponen un modelo basado en la Programación Orientada a Objetos (POO) que requiere la formación de estrategias heurísticas particulares, facilitando la búsqueda de la solución a un problema. Sin desdeñar esta propuesta, es criticable que se dirija a las generalidades de la POO lo que implica un tiempo considerable para su estudio, en detrimento del aprendizaje y desarrollo de habilidades inherentes a la algoritmización.

Dentro de los partidarios del segundo enfoque didáctico, anteriormente citado, se destaca la metodología de A. Ferreira y G. Rojo (2005), sustentada en la ejecución de varias fases (análisis, diseño, implementación y prueba) para resolver un problema en un lenguaje algorítmico de carácter general, el que luego se traduce a uno de alto nivel. Sin embargo, aun cuando esta propuesta resulta válida, en tanto reconoce la algoritmización como esencia del proceso de programación, no profundiza en las particularidades de los procesos que intervienen en la lógica didáctica de la misma. Esto se debe, fundamentalmente, a que no se favorece el componente lógico – matemático necesario en un lenguaje algorítmico estructurado, que posibilite una interpretación y modelación matemática, previa a la implementación en un lenguaje de programación.

También se destaca la propuesta de los investigadores franceses N. Guibert, L. Guittet y P. Girard (2006), consistente en una metodología que describe cuatro momentos principales del proceso de resolución de un problema de programación utilizando pseudocódigos en un software denominado “MELBA” y la de B. Faouzia y H. Mostafa (2006) de un software llamado “EasyAlgo”, que permite algoritmizar usando pseudocódigos. Si bien estas propuestas reconocen y abordan la algoritmización como proceso esencial dentro de la enseñanza de la programación, es criticable el hecho de que no brindan elementos didácticos que permitan revelar las relaciones que se establecen entre las etapas del mencionado proceso, lo que potenciaría su aprendizaje. De aquí que, aun cuando aportan instrumentos didácticos, no siempre se alcancen las habilidades esperadas.

Por su parte los investigadores argentinos N. Arellano y otros (2014), proponen una metodología para la enseñanza de la programación a estudiantes noveles, usando varios software de apoyo para introducir al estudiante en la noción de algoritmo, complementar gráficamente su resolución, ejecutarlo y depurarlo. Esta

estrategia favorece la creación de un pensamiento computacional, pero se centra en el uso de herramientas tecnológicas, sin utilizar algún instrumento didáctico, contentivo de acciones que estructuren e integren el uso de los citados software, para orientar el proceso de enseñanza – aprendizaje de la programación.

Todo lo anterior permite proponer como **objetivo**, la elaboración de un sistema de procedimientos didácticos para la algoritmización computacional, sustentado en un modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional, que permita potenciar el desarrollo de la enseñanza y el aprendizaje de dicho proceso.

Consecuentemente se asume la necesidad de revelar en un movimiento único la lógica didáctica de la algoritmización computacional en la resolución de problemas de programación, como eje integrador que favorezca la comprensión y modelación matemático – computacional previa al proceso de implementación en un lenguaje de programación, lo que se constituye en **orientación epistémica** de esta investigación.

Por lo que se define como **hipótesis,** que si se aplica un sistema de procedimientos didácticos, sustentado en un modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional, que tome en cuenta la contradicción dialéctica que se manifiesta entre una representación matemática problematizada y su sistematización algorítmica, dinamizadora del proceso de resolución de dichos problemas, se favorece el logro de resultados significativos en la apropiación y aplicación de contenidos de programación a la solución de problemas computacionales.

Una vez definido el camino hipotético que dará solución al problema científico de la presente investigación, se precisan las siguientes **tareas científicas** a desarrollar:

1. Fundamentar epistemológicamente el proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional.
2. Determinar las tendencias históricas del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional.

3. Caracterizar el estado actual del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional en las carreras de ciencias computacionales de la Universidad de Oriente.
4. Elaborar un modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional.
5. Elaborar un sistema de procedimientos didácticos para la algoritmización computacional.
6. Valorar la factibilidad y pertinencia científico – metodológica de los principales resultados de la investigación a partir de la realización de talleres de socialización con especialistas.
7. Ejemplificar el sistema de procedimientos didácticos que se propone.
8. Aplicar parcialmente el sistema de procedimientos didácticos en el primer año de la carrera de Ingeniería en Telecomunicaciones y Electrónica de la Universidad de Oriente.

Como **métodos, enfoques y técnicas** de investigación se utilizaron:

- Histórico – lógico en la caracterización histórica del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional y en todo el proceso investigativo.
- Análisis – síntesis: transitó por todo el proceso de investigación científica.
- Holístico – dialéctico para el diseño del modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional y en toda la lógica del proceso investigativo.
- Sistémico – estructural – funcional en la elaboración del sistema de procedimientos didácticos.
- Métodos y técnicas empíricas, para la caracterización del estado actual del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional y su dinámica de algoritmización, así como para la corroboración de los resultados científicos alcanzados.
- Métodos cuantitativos para obtener la información necesaria y procesarla.
- Enfoque hermenéutico – dialéctico, que facilitó una lógica científica para la comprensión, explicación e interpretación del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional y su dinámica de algoritmización.

El **aporte teórico** de la investigación consiste en haber revelado la doble modelación, matemática y computacional, que se da en el proceso de resolución de un problema de programación computacional, la cual se materializó en la fundamentación de un modelo de la dinámica lógico – algorítmica del citado proceso.

El **aporte práctico** es un sistema de procedimientos didácticos para la algoritmización computacional, que concreta e instrumenta el modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional.

El **impacto social** de la investigación radica en la posibilidad de favorecer el logro de resultados superiores en el proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional, a partir de potenciar en el estudiante la formación de un pensamiento algorítmico computacional que le permita elevar la eficacia y eficiencia resolutora, perfeccionando su formación profesional.

La **novedad científica** consistió en haber revelado y fundamentado una lógica integradora entre la representación matemática de la situación problémica y su generalización pseudocodificada, como condición imprescindible para el desarrollo de un pensamiento algorítmico computacional; revelándose la doble modelación, matemática y computacional, que debe experimentar una situación problémica en el camino algorítmico que conduce a su solución.

Aquí se reconoce la lógica integradora como un eje orientador hacia la generalización pseudocodificada, que al integrar los dos estadios resultantes de movimientos epistémicos esenciales de la modelación, permite establecer una relación de correspondencia entre los objetos matemáticos que representan una situación problémica y los objetos computacionales que la generalizan, todo lo cual se desarrolla mediante un movimiento en espiral que va cambiando de dirección a cada paso y vuelve a la misma posición, pero elevándose de nivel, favoreciendo así la transformación y la interdependencia de dichos objetos, en aras de resolver la situación problémica, justamente estos son los elementos que superan pedagógicamente a los modelos existentes.

CAPÍTULO I: CARACTERIZACIÓN DEL PROCESO DE ENSEÑANZA – APRENDIZAJE DE LA RESOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN COMPUTACIONAL

Introducción

En este capítulo se presentan aspectos esenciales del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional, con énfasis en la dinámica de la algoritmización que en el mismo se lleva a cabo. Se fundamenta el mencionado proceso desde un estudio epistemológico que permite precisar sus principales referentes y analizar categorías y relaciones esenciales que se dan en el mismo. De igual forma, se revela el comportamiento histórico de dicho proceso, lo que ha permitido precisar sus principales tendencias. Finalmente, se realiza una caracterización de su estado actual, haciendo énfasis en la dinámica del proceso de algoritmización computacional en las carreras de ciencias computacionales de la Universidad de Oriente, lo que ha propiciado la profundización en sus limitaciones fundamentales.

1.1 Fundamentación epistemológica del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional

La programación computacional es aquella parte de las ciencias computacionales que proporciona las herramientas necesarias para establecer la secuencia de instrucciones que deben ser aplicadas, empleando un lenguaje específico, para que el ordenador ejecute una tarea, previo análisis de las condiciones y requerimientos de la misma (ACM, 2013).

Consecuentemente, la actividad de *programar* requerirá del establecimiento de un conjunto de instrucciones lógicamente ordenadas que controlen el comportamiento físico y lógico de una computadora, para expresar la solución computacional de alguna situación problémica, lo que implica que se deban formar habilidades en los

estudiantes para diseñar, implementar y refinar el código fuente de los programas computacionales. El mencionado código puede ser escrito en un lenguaje natural ó en un lenguaje de programación determinado, requiriendo de conocimientos de algoritmos matemáticos y computacionales, así como de la lógica formal, lo que permitirá diseñar e implementar programas computacionales que cumplan los requerimientos de determinada situación problémica (Salgado, A., Alonso, I. y Gorina, A., 2014a).

Ahora bien, en la presente investigación se reconoce la lógica formal según E. Bueno (1976), quien la define como aquella ciencia que se basa en leyes, modalidades y formas del conocimiento científico, que posee un carácter formal y se enfoca hacia el estudio de alternativas de inferencia válidas; según la cual se pueden estudiar los métodos y principios adecuados para identificar el razonamiento correcto frente al que no lo es.

También se consideran los progresos de dicha ciencia, la que en sus inicios se basó en los postulados aristotélicos que sólo consideraban las alternativas de verdadero y falso, pero que con el desarrollo de la Matemática han llegado al reconocimiento de múltiples alternativas, que se traducen en una pluralidad de formas de pensar, de conocer y de explicar la realidad, el mundo y los fenómenos. De aquí que para la presente investigación se asuma esta lógica polivalente entre las tantas alternativas que existen para llevar a cabo el proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional.

Por consiguiente, hablar de la programación computacional implica reconocerla como un proceso complejo y creativo, el que ha sido abordado desde una diversidad de paradigmas que han generado numerosas propuestas y discusiones, en la búsqueda de una forma óptima de enseñar a programar, que permita al alumno el desarrollo de sus potencialidades para utilizar un conjunto de abstracciones, interrelacionadas para la resolución de problemas (Dillon, E., Anderson, M. y Brown, M., 2014; Kinnunen, P. y Simon, B., 2012).

Es así que, por sus beneficiosos efectos en el desarrollo de las capacidades cognitivas de los estudiantes, la programación suscita interés psicopedagógico al contribuir al desarrollo de su actividad intelectual, en tanto les permite establecer planificaciones y estrategias, construir algoritmos, estructurar instrucciones, analizar y

comprender los programas propios o escritos por otros, aprender la sintaxis de los distintos lenguajes de programación y comparar recursos usados por programadores expertos y principiantes (Miños, A. M, 2015; Kurkovsky, S., 2013; Fariñas, J. L., 2009). Es así que todos estos elementos a decir de J. M. Wing (2006) se dirigen a potenciar la formación de un *pensamiento computacional*, considerado en esta investigación como el proceso mental, sistemático e integrador, que desarrolla un estudiante para llegar a un resultado computacional pertinente en la resolución de una situación problémica, a partir de su cultura computacional y del razonamiento que lleva a cabo en el marco de una lógica formal (Salgado, A. y otros, 2013a).

Cabe precisar que por *situación problémica* se entiende una determinada situación en la cual existen nexos, relaciones, cualidades de y entre los objetos que intervienen en ella, que no son accesibles directa o inmediatamente al individuo que la aborda (Santiesteban, Y., 2013; Rodríguez, M., 2013; Alonso, I., 2001; Labarrere, A., 1994; Torres, P., 1992). De aquí que se constituya en un reto para desarrollar la comprensión en los alumnos, los cuales en busca de su solución deben poner en práctica sus conocimientos, ingenio y destrezas, convirtiéndose así en productores de un nuevo nivel de aprendizaje con resultados significativos.

Ahora bien, la complejidad que caracteriza a la programación computacional hace que al llevarla a cabo se confronten serias dificultades, siendo una de las principales, la falta de éxito que tienen los estudiantes en el análisis y solución de situaciones problémicas empleando objetos computacionales (Ramírez, R. V., 1991).

A los efectos de la presente investigación la situación problémica es interpretada como una situación conformada por objetos reales, matemáticos o computacionales, que han sido didácticamente tratados por el profesor para hacerlos asequibles, en dependencia del nivel de profundidad requerido por los estudiantes que se inician en el aprendizaje de la resolución de problemas de programación computacional. La misma adquiere la connotación de *problema de programación computacional* en la medida en que el estudiante asume su resolución y encamina sus esfuerzos a la interpretación matemática y computacional de la misma (Salgado, A., Alonso, I. y Gorina, A., 2014).

De aquí que se considere que *resolver un problema de programación computacional* implique el establecimiento de una sucesión de pasos elementales, cada uno de los cuales genera un conocimiento nuevo que se obtiene como inferencia lógica a partir de los conocimientos y experiencias del individuo y de las condiciones del problema o consecuencias derivadas de éstas en pasos anteriores. La conjunción de estos pasos permitirá entonces fundamentar la exigencia de dicho problema y desarrollar un programa que lo resuelva mediante un ordenador (Salgado, A. y otros, 2013a).

En tal sentido G. Polya (1965) propone un modelo de resolución de problemas que se concreta en las siguientes etapas: comprender el problema, concebir un plan de solución, ejecutar el plan, y examinar la solución obtenida. Este modelo, aun cuando está pensado para problemas matemáticos y no se constituye en procedimientos estrictos que puedan asegurar el éxito al resolver los mismos, deviene en alternativa viable para una orientación general, reconocida por su representatividad en la didáctica de la programación computacional (Arellano, J. J. y otros, 2012).

En este modelo resulta de especial importancia la etapa de *comprensión*, la que en muchos casos es subestimada por los estudiantes, motivo por el cual resultan comunes los errores durante el proceso de resolución de una situación problémica. Precisamente ha sido reconocida por numerosos didactas la importancia de tener claridad sobre el problema que se quiere resolver antes de proceder a su solución, conclusión a la que han arribado a partir de las observaciones hechas al desempeño de los alumnos, que muchas veces fallan porque leen el problema rápido, sin poner atención a los detalles y sin preocuparse por comprenderlo completamente; lo que los lleva a resolver un problema que es diferente del que se les plantea (Alonso, I., 2001; Santos, L.,1995; Pozo, J. I.,1994; Whimbey, A. y Lochhead, J., 1993; Polya, G.,1965).

La *comprensión* ha sido definida por R. Wiltrock (1990: 7) como "…una representación estructural o conceptualmente ordenada de las relaciones entre las partes de la información que se debe aprender y entre esa información y esas ideas y nuestra base de conocimientos y experiencias". De la anterior definición se

puede inferir, a decir de I. Alonso (2001: 4) que "…la comprensión de una situación problémica depende, ante todo, de la representación que de la misma se haga la persona que trata de resolverla, y por tanto esta última juega un papel muy importante en su resolución, ya que el proceso comienza justamente con la concepción de una *representación* de la mencionada situación".

Así, para la concepción de adecuadas representaciones matemáticas de la situación problémica bajo estudio, será necesario llevar a cabo un apropiado proceso de análisis de la misma, que conduzca a su correcta interpretación. Considerando la habilidad de *interpretar* como el proceso mediante el cual se captura información de la citada situación problémica en un contexto determinado, atribuyéndole significado a los objetos que conforman dicha situación, de modo que adquieran sentido en función de la base de conocimientos matemático – computacionales y de las experiencias previas del individuo (González, M. C. y Paniagua, J. G., 2010; Alberto, M., 2010; Niño, R., 2005; Alonso, I., 2001; Delgado, J. R., 2000). Además permite adaptar a un marco matemático – computacional el lenguaje de las otras disciplinas de estudio, para luego traducirlo de nuevo al lenguaje del usuario.

Ahora bien, un referente para sustentar la representación lo propicia la Didáctica de la Matemática cuando postula que, mediante el trabajo con las representaciones los estudiantes asignan significados y comprenden las estructuras matemáticas, sin lo cual no es posible la resolución de problemas. Las *representaciones matemáticas* se entienden, en sentido amplio, como todas aquellas herramientas –signos o gráficos- que hacen presentes los conceptos y procedimientos matemáticos y con los cuales los estudiantes abordan e interactúan con el conocimiento de esta ciencia, es decir, registran, asimilan y comunican su conocimiento sobre la Matemática (Alonso, I., 2001; Duval, R., 1993 y 1999; Rico, L., Castro, E. y Romero, I., 1996).

En cuanto a la segunda y tercera etapa del modelo de G. Polya, concepción y ejecución de un plan de solución, los investigadores A. Whimbey y J. Lochhead (1993) mencionan errores comunes de los estudiantes al resolver problemas, ya que en ocasiones no tienen claro el procedimiento a utilizar y comienzan a trabajar

con el que más se le parece, sin razonar sobre su conveniencia, ni estimar los posibles resultados que deben obtener, resolviendo el problema de manera mecánica, sin reflexionar mucho sobre la eficacia de lo que está haciendo. Ello lleva a que, en esos casos, se ejecuten planes de solución que no están precedidos de un diseño pertinente y veraz, que garantice una lógica coherente para la continuidad del proceso.

Consecuentemente, en la enseñanza de la resolución de problemas de programación computacional es imprescindible insistir en la realización de acciones para lograr una correcta comprensión e interpretación de la situación problémica inicial, así como la identificación de los objetos que intervienen en la situación problémica, entendiendo por *identificar* "…distinguir el objeto de estudio… sobre la base de sus rasgos esenciales. Determinar si el objeto pertenece a una determinada clase de objetos que presentan ciertas características distintivas" (Delgado, J. R., 2000: 4).

Todo lo anterior posibilitará el logro de una representación matemática pertinente y garantizará la validez de la correspondiente representación computacional, que es la que permite concretar la solución requerida a partir de objetos y relaciones computacionales. Sin embargo, en la actualidad los cursos de Programación muy rara vez hacen énfasis en estos aspectos, centrándose directamente en la representación computacional de la situación problémica, lo que explica en gran medida el poco éxito que en la programación tienen los estudiantes (Salgado, A. y otros, 2013; Arellano, J. J. y O. S. Nieva, 2009).

Otro proceso importante en la resolución de problemas es el correspondiente a la cuarta etapa del modelo de G. Polya, referido al examen de la solución obtenida, el que a los efectos de la programación computacional es equivalente a la validación de dicha solución. Esta *validación* se concibe como la verificación de la solución computacional en función de las condiciones y exigencias originales de un problema de programación (Arellano, N. y otros, 2014; Salgado, A. y otros, 2013a; Arellano, J. J. y otros, 2012).

Ahora bien, como parte de la validación se requiere el desarrollo de un proceso de *valoración*, entendida esta como un concepto al cual se llega a través de un proceso de razonamiento, a partir de toda una serie de

juicios sobre los diversos aspectos que son evaluados (Fuentes, H. C., Montoya, J. y Fuentes, L., 2012). Así, la valoración además de constituir el resultado de un proceso de validación es punto de partida o juicio para un razonamiento posterior que conduzca a nuevas valoraciones.

Los autores A. Whimbey y J. Lochhead (1993) mencionan dentro de los errores más comunes, el consistente en llegar a la solución y no comprobar que sea la correcta. Esto se manifiesta en que generalmente se prueba el programa con los datos que aparecen como ejemplo en el planteamiento del problema y, si funciona adecuadamente, se supone que el programa ya es correcto. Sin embargo, en numerosas ocasiones, al probarlo con datos reales, el programa no se ejecuta correctamente o no proporciona el resultado esperado.

El proceso de validación es un aspecto clave en la resolución de un problema computacional, pues un programa computacional se constituye en solución del mismo cuando se ha logrado demostrar su validez con respecto a todos los posibles rangos de datos entrantes y cumple con las exigencias y condiciones de dicho problema (Fergusson, E. M. y otros., 2015; Wengrowicz, N., 2014). Sin embargo, es usual que los estudiantes se interesen poco por realizar una validación exhaustiva, lo que da cuenta de la necesidad de seguir perfeccionando el proceso de validación como etapa esencial de la sistematización de la programación, que conduce a que se logren cualidades tan importantes como la pertinencia, precisión y exactitud.

A partir de las ideas anteriores se puede inferir la gran complejidad que entraña el proceso de resolución de un problema de programación computacional, el que demanda del desarrollo de habilidades para analizar, comprender, interpretar y modelar la situación problémica a resolver, creando algoritmos de solución eficientes para luego codificarlos siguiendo las reglas sintácticas y semánticas de un determinado lenguaje de programación y comprobar su eficacia para resolver la situación problémica original.

Cabe precisar que en la presente investigación se asume como *eficacia computacional* la característica que debe cumplir el algoritmo desarrollado, en relación a su capacidad de lograr el efecto que se desea o se espera, es decir, que resuelva completamente la situación problémica planteada, cumpliendo con la

intencionalidad deseada y respondiendo a las exigencias de la mencionada situación, a partir de las condiciones iniciales que esta brinda.

Por otro lado, la *eficiencia computacional* es entendida como la característica que debe cumplir el algoritmo, relativa a la capacidad de lograr el objetivo para el que se ha diseñado, con el mínimo de recursos computacionales y en el menor tiempo posible. Estos recursos se refieren a las estructuras matemáticas (gráficos, tablas, ecuaciones y funciones, etc.) y computacionales (for, if, while, do while, entre otras) que facilitan realizar las representaciones matemáticas y computacionales. Logrando así dar solución a una determinada situación problémica en un tiempo razonable.

Por otro lado, debe destacarse el importante papel de la *modelación* en la presente investigación, la que ha sido estudiada por numerosos científicos, dentro de los que se puede citar a O. Planchart (2005), M. Salett y N. Hein (2004), E. Castro y E. Castro (1997). En esta dirección se asume lo postulado por J. R. Delgado (2000: 7) al plantear que la habilidad modelar "…se manifiesta en tres niveles fundamentales: de selección, cuando es conocido el objeto matemático que se utiliza para modelar; de adaptación, cuando no se conoce de antemano el modelo matemático y se adapta o usa en condiciones nuevas y de creación, cuando no existe un modelo matemático ni es posible adaptar alguno existente debido a las exigencias del objeto a modelar".

A criterio del postulante, la modelación cobra especial relevancia en el proceso de resolución de los problemas de programación computacional, al requerirse dos modelaciones en las que los objetos representantes son, en primera instancia de naturaleza matemática y en segunda, computacional.

Así, la primera modelación asocia objetos matemáticos a todo objeto real o ideal, los que representan determinados comportamientos, relaciones o características suyas (Delgado, J. R., 2000), mientras que la segunda, más esencial aún que la primera, asocia *estructuras computacionales* (entidades computacionales como if, for, while, etc.) a los objetos matemáticos obtenidos, las que permiten la ejecución de iteraciones o

repeticiones dentro de un programa. Las citadas estructuras conforman el pseudocódigo, mediante el cual se logra una representación generalizada de la primera modelación hecha (Salgado, A. y otros, 2013b).

De lo anterior se deriva la necesidad de dedicar especial atención al desarrollo de la habilidad *modelar* en el proceso de enseñanza – aprendizaje de las asignaturas computacionales, especialmente de aquellas cuyos objetivos estén encaminados a enseñar a programar.

En esta dirección resultan interesantes los resultados obtenidos por los investigadores L. Zamora, P. Orús y J. R. Díaz, (2008 y 2010) en las carreras de Licenciatura en Ciencia de la Computación y Licenciatura en Matemática en la Universidad de Oriente, Cuba. Dichos autores profundizaron en el aprendizaje de los estudiantes de ambas carreras, pudiendo concluir que cuando un estudiante de primer año logra apropiarse de los contenidos de las asignaturas Análisis Matemático y Álgebra, tiene grandes probabilidades de aprobar la asignatura de Programación y que el estudiante que aprueba Programación tiene una probabilidad de al menos 95 % de aprobar todas las asignaturas del año. Estos resultados pueden tomarse como sustento empírico de la idea del postulante sobre la necesidad de enseñar a modelar matemáticamente la situación problémica, para luego realizar su modelación computacional.

En este mismo orden de ideas las investigadoras cubanas I. Urrutia y V. Álvarez (2009), de la Universidad de La Habana y A. Vargas, O. L. Pérez y R. Blanco (2014) de la Universidad de las Ciencias Informáticas, analizan cómo desde las Disciplinas Análisis Matemático (en el primer caso) y Algebra Lineal (en el segundo), se puede contribuir a formar habilidades básicas en los estudiantes que están aprendiendo a programar, las que están encaminadas a favorecer la modelación a partir del empleo de sólidos conocimientos matemáticos que le permiten realizar las tareas de programación computacional.

Asimismo los investigadores cubanos Y. Soler y otros (2008) argumentan que debido a los conocimientos adquiridos en las asignaturas de Álgebra y Matemática, los alumnos son capaces de modelar tomando como

base objetos matemáticos tales como números enteros, complejos, matrices y operaciones sobre ellos, lo que sirve de base para la obtención de nuevos modelos de naturaleza computacional.

También autores como, O. Yasar (2013), plantean la conveniencia de modelar matemáticamente el problema que se trata de resolver, pues cuando ya se cuenta con un modelo matemático adecuado al problema, el objetivo consiste en hallar una solución en forma de algoritmo, para lo cual se puede usar un pseudolenguaje abstrayéndose de la sintaxis de los lenguajes formales.

Teniendo en cuenta todo lo anterior puede concluirse que es una necesidad la introducción de una doble modelación en el proceso de resolución de problemas de programación computacional y que la modelación matemática es de vital importancia para el logro de la misma pues permite:

- Una representación más precisa, analítica, abstracta, sintética y generalizada de la situación problémica a partir de objetos matemáticos, al potenciarse los procesos lógicos del pensamiento (análisis, síntesis, inducción, deducción, generalización, abstracción, concreción).
- La obtención de una posible solución matemática de la situación problémica, expresada en lenguaje matemático sintético, que es más isomorfa a las estructuras computacionales existentes para llevar a cabo la modelación computacional que el propio lenguaje natural utilizado para describir dicha situación.
- Un primer estadio en el procesamiento de la información de la situación problémica, que reduce su volumen de información, lo que posibilita que la misma sea más abarcable, tratable y orientada a los efectos de la modelación computacional; favoreciendo así los procesos de comprensión, representación, identificación, estructuración, jerarquización, control, validación y generalización computacional.

De esta forma el proceso de resolución de problemas de programación computacional puede verse como la evolución, en el pensamiento del estudiante, de las diferentes representaciones del problema (matemáticas y computacionales) (Salgado, A., Gorina, A. y Alonso, I., 2013).

En este orden de ideas es esencial precisar que en el tránsito entre las dos modelaciones correspondientes al proceso de resolución de un problema de programación computacional emerge como habilidad esencial la

algoritmización computacional, que consiste en plantear una sucesión de operaciones computacionales mediante pseudocódigos, diagramas de flujo o diagramas de Nassi – Schneiderman, tomando como base las relaciones develadas entre los objetos matemáticos, encaminadas a la solución de una situación problémica (Pinales, F. J. y Velázquez, C. E., 2014; Salgado, A. y otros, 2013a; Reynolds, C. y Tymann, P., 2008).

Consecuentemente, y siendo coherente con la lógica del presente discurso, es necesario precisar que el término "algoritmo", proviene del nombre del matemático árabe Muhammed ibn Musa al-Khwarizmi (Van Dalen, B., 1996); denominándose algoritmo a un grupo finito de operaciones organizadas de manera lógica y ordenada, que permite solucionar un determinado problema (Skiena, S. S., 2008).

Para el caso particular de las ciencias computacionales, un algoritmo es una serie de instrucciones o reglas (matemáticas y computacionales) lógicamente establecidas, las que por medio de una sucesión de pasos lógicos transforman algún valor o conjunto de valores de entrada en uno o varios valores de salida, arribando a un resultado computacional que soluciona una situación problémica (Salgado, A. y otros, 2013a).

Aquí cabe precisar que el *pseudocódigo* es definido como una estructura computacional que es expresada en lenguaje natural y posibilita formular un algoritmo o programa sin necesidad de usar un lenguaje de programación específico (Skiena, S. S., 2008; Ferreira, A. y Rojo, G., 2005).

Así el empleo de pseudocódigos permite una generalización de la representación matemática, dando lugar a una nueva representación de naturaleza computacional. Esta *generalización* es asumida, en términos didácticos generales, como la abstracción de lo que es común a las categorías internas del objeto, pero en su relación con aquellas de su existencia ontológica y epistémica, logrando aportar consideraciones conceptuales que les son comunes y que reflejan el sistema de relaciones que tipifican de manera generalizadora dicho objeto. De modo que los resultados obtenidos, aun cuando hayan sido producidos en un contexto concreto determinado, tienen posibilidades de ampliarse a otros contextos de naturaleza similar para lograr transformaciones cualitativas (Matos, E. C. y Cruz, L., 2011; Fuentes, H. C., 2009).

Ahora bien, la generalización matemática implica la abstracción de los elementos esenciales de la situación problémica, en su relación con los objetos, características y relaciones matemáticas que permitan una representación que la tipifique y eleve a un primer nivel de generalización. Esto permitirá su posterior representación mediante pseudocódigos, con lo que se podrá lograr un segundo nivel de generalización, más esencial (Salgado, A. y otros, 2013; Brennan, K. y Resnick, M., 2013). El proceso descrito es reiterativo, pues es susceptible a ser perfeccionado sistemáticamente, lográndose su paulatina automatización. Esto, en opinión del postulante, connota la doble modelación que debe caracterizar a la resolución de un problema de programación computacional.

Así la algoritmización tiene una doble significación: cognoscitiva y metodológica. Cognoscitiva porque el algoritmo se constituye en soporte teórico materializado que expresa la secuencia lógica y estricta de la modelación realizada. Metodológica, porque la sucesión de operaciones contenidas en el algoritmo, puede servir como base de orientación para realizar la acción o implementación (Delgado, J. R., 2000).

El proceso de algoritmización descrito debe conducir a la obtención de un algoritmo con las siguientes características (Cormen, T. H. y otros, 2009; Skiena, S. S., 2008):

- Precisión: debe indicar el orden o jerarquía de realización de cada acción, de forma clara y sin ambigüedades (integración de estructuras), en lo que influye directamente la identificación adecuada de las estructuras computacionales a utilizar (for, while, if, etc.).
- Concreción: en el sentido de contener sólo el número de pasos precisos para llegar a la solución.
- Repetitividad: debe alcanzar siempre los mismos resultados para una misma entrada, independientemente del momento de ejecución.
- Validez: debe controlar que el algoritmo esté sintácticamente correcto (que las estructuras computacionales elegidas estén correctamente escritas) y en segundo lugar que resuelva el problema completamente, cumpliendo con la intencionalidad deseada (validación semántica).

- Finitud: debe terminar en algún momento.
- Eficiencia: al dar una solución en un tiempo razonable.
- Optimización: da respuesta a la cuestión de si el algoritmo diseñado para resolver el problema es el mejor.

En este sentido y como norma general, será conveniente tener en cuenta que suele ser mejor un algoritmo sencillo que no uno complejo, siempre que el primero no sea extremadamente ineficiente.

Por otro lado, todo algoritmo contempla las tres partes fundamentales de una solución informática: a) entrada, que es la información dada al algoritmo, b) procesamiento o cálculos necesarios para encontrar la solución del problema y c) salida, que son los resultados finales de los cálculos.

Es así que el algoritmo debe describir una transformación de los datos de entrada a objetos matemáticos y de estos últimos a objetos computacionales para obtener los datos de salida, permitiendo realizar su *interpretación* mediante un análisis posterior a dicha salida (Arellano, J. J. y O. S. Nieva, 2009).

Ahora bien, con independencia de la importancia que tiene la resolución de problemas de programación computacional, en la práctica siguen existiendo dificultades con la formación de habilidades para llevarla a cabo, lo que ha dirigido la atención de los docentes e investigadores hacia su esencialidad didáctica, a partir de reconocer la existencia de diferentes enfoques para la enseñanza de la programación, los que son valorados en la presente investigación, por ser especialmente significativos para la resolución de problemas.

Entre los aportes hechos por los investigadores que han estudiado los enfoques más empleados en la enseñanza de la programación, sobresale la clasificación propuesta por C. Espósito y otros (2001), quienes reconocen el del manual, proyecto, problema base, modelo, problémico y algorítmico.

Así, desde la perspectiva del enfoque del manual o instruccionista, la enseñanza de la programación se caracteriza por poner énfasis en los elementos del lenguaje, y no en los procesos de búsqueda de solución de problemas. Por consiguiente, su aplicación es válida sólo cuando el estudiante ha adquirido conocimientos

sobre las estructuras computacionales y su uso, por lo que se estudian los elementos, la sintaxis, la semántica (objetivo y significado) y ejemplos de su uso (Käasboll, J., 1998; Caignaert, C., 1988).

En opinión de este autor el citado enfoque no es totalmente viable pues, al poner el énfasis en los elementos del lenguaje, descuida la formación de habilidades básicas para programar, tales como, el diseño de los algoritmos a utilizar como paso previo a la implementación de un programa mediante un lenguaje de programación. Además no potencia etapas previas a la implementación, como la comprensión, interpretación y representación de la situación problémica. Esto conlleva a que los programadores no expertos dominen bien la sintaxis de los lenguajes pero no sepan diseñar programas que cumplan con la funcionalidad deseada.

Por su parte, el enfoque algorítmico se dirige al desarrollo de habilidades para la resolución de problemas en los procesos de búsqueda, por lo que hace énfasis en el desarrollo de procedimientos algorítmicos y heurísticos para resolver problemas por medios informáticos (Arellano, J. J. y O. S. Nieva, 2009; Faouzia, B. y Mostafa, H., 2006, 2007; Oviedo, M. y Ortiz, F. G., 2002). Por tanto, los contenidos referidos a lenguajes o software para usos específicos pasan a ocupar un segundo plano, al buscarse la solución de un problema aplicando recursos heurísticos (reglas, estrategias, principios) y algorítmicos (procedimientos básicos ya conocidos); es decir, se modela la solución mediante una descripción algorítmica (Arellano, J. J. y otros, 2012; Martínez, S. y Fariñas, J. L., 2012; Guilbert, N., Guittet, L. y Girard, P., 2006; Levine, G., 2001).

El postulante considera que este enfoque, al usar pseudocódigo, evita muchas ambigüedades del lenguaje natural. Dichas expresiones son formas más estructuradas para representar algoritmos; no obstante, se mantienen independientes de un lenguaje de programación específico. La descripción de un algoritmo usualmente se hace en dos niveles, primero una descripción de alto nivel donde se establece el problema, se selecciona un modelo matemático y se explica el algoritmo de manera verbal, posiblemente con ilustraciones y omitiendo detalles, esto permite al programador novel tener una primera aproximación de lo que será una solución formal del problema.

En el segundo nivel se hace una descripción formal, usándose el pseudocódigo para describir la secuencia de pasos que conducen a la solución, lo que permite que el estudiante se oriente en relación con las estructuras computacionales que debe utilizar y la forma de hacerlo. A través de este enfoque se logra formar en el estudiante una lógica de algoritmización al tener que pasar por distintos niveles de creación o diseño, cada uno más complejo que el anterior. Sin embargo este enfoque también presenta limitaciones, pues a pesar de centrarse en el proceso de algoritmización, no garantiza que el mismo se sustente en una modelación matemática previa, resultante de los procesos de comprensión, interpretación y representación de la situación problémica, por lo que se limita la formación del pensamiento algorítmico en el estudiante.

El enfoque del proyecto, como tercer exponente, tiene como propósito esencial motivar la enseñanza de los contenidos de la programación computacional a través del planteamiento y ejecución de un proyecto. Se caracteriza por la división del mismo en subproyectos más sencillos que motiven el aprendizaje del nuevo contenido informático (Murillo, M., 2006). Así cada fase debe motivar la obtención de nuevos conocimientos informáticos, según la vía lógica elegida, seguido de la realización de acciones de fijación inmediata, teniendo en cuenta los pasos esenciales del procedimiento. Finalmente se aplica el nuevo conocimiento o parte del mismo a la solución de la correspondiente fase del proyecto (González, W., 2001).

Si bien este enfoque propicia la motivación del estudiante hacia la asignatura de Programación, tiene como desventaja que el desarrollo de los programas se hace directamente en un lenguaje de alto nivel, lo que implica que sea necesario aprender la sintaxis, las estructuras y las herramientas computacionales propias del lenguaje. Por otro lado, al no hacerse énfasis en la algoritmización, no se contribuye suficientemente a la formación de habilidades de programación que son esenciales, tales como la representación y la generalización, que permiten arribar a estadios superiores en la solución de la situación problémica.

A diferencia del anterior, el enfoque del problema base pretende motivar la enseñanza de los contenidos de la programación computacional a través de diferentes modificaciones que progresivamente se van formulando al

planteamiento inicial de un problema, procediendo de forma inversa al enfoque del proyecto, en tanto, cada modificación planteada al problema inicial es un recurso que debe motivar la necesidad del nuevo conocimiento (Chesñevar, C. I., 2001). Desde esta perspectiva didáctica se facilita la asimilación de lo nuevo, ya que se parte siempre de lo conocido. Ello permite que el problema inicial se vaya transformando, adquiriendo niveles de exigencias superiores en la medida en que se dominan los conocimientos necesarios y se aplica el nuevo conocimiento a la solución de la modificación correspondiente al problema base.

Este enfoque constituye una alternativa más efectiva que la del enfoque del proyecto, sin embargo no trasciende el hecho de que se le presenta al estudiante un problema que aunque puede ser sencillo de resolver, no fomenta suficientemente las habilidades de comprensión, análisis, representación, modelación e identificación de estructuras computacionales. El mismo es conveniente cuando se desea enseñar al estudiante a trabajar con aplicaciones computacionales como Microsoft Office, Statistica, entre otros.

Desde un tratamiento didáctico más racional, el enfoque del modelo procura simular fenómenos o procedimientos como un medio para inferir los elementos esenciales del nuevo conocimiento informático objeto de estudio. Este se caracteriza por el uso de un programa que realiza dicha simulación, por lo que el modelo como recurso didáctico debe estar elaborado de forma tal que muestre, o se puedan inferir con claridad, los elementos esenciales del objeto modelado. Se simula el fenómeno o proceso haciendo uso del medio, lo que permite inferir los elementos esenciales del nuevo conocimiento informático objeto de estudio y formalizar el concepto o procedimiento (Espósito, C., 1995).

Si bien este enfoque contribuye a motivar a los estudiantes a través del uso de software simuladores, mapas conceptuales, libros electrónicos etc., tiene como principal limitación que en dependencia de lo bien representado que esté el modelo escogido, de lo que se quiere enseñar y de la experiencia del profesor, se lograrán los resultados deseados. Además, el mismo no propicia el pensamiento algorítmico, ya que los estudiantes se centran en tratar de simular un proceso o resolver un problema sin haber adquirido suficientes

habilidades básicas de representación matemático – algorítmica del mismo, a partir del uso de un lenguaje como puede ser el pseudocódigo, los diagramas de flujo o los diagramas de Nassi – Schneiderman.

Por último, el enfoque problémico tiene en su base esencial la resolución de problemas. Se sustenta en la enseñanza problémica, aunque no desde una aplicación estricta de la misma, por lo que el énfasis principal es la creación de situaciones problémicas que favorezcan la necesidad del nuevo conocimiento informático que debe ser objeto de estudio, a partir de lograrse una motivación efectiva, según la vía lógica elegida y su aplicación a la solución del problema bajo análisis (Neyret, R., 2005; Martínez, M., 1999).

Este enfoque tiene la ventaja de facilitar la motivación del estudiante, siempre que se escoja adecuadamente la situación problémica a resolver; pudiendo vincularse con el enfoque algorítmico, planteándole al estudiante situaciones que demanden el uso de pseudocódigos cada vez más complejos. Es preciso señalar que es muy probable que no se obtengan buenos resultados si se le exige al estudiante que proponga soluciones usando directamente la sintaxis de un lenguaje, pues no se facilita la formación de habilidades necesarias para desarrollar el pensamiento lógico, matemático y algorítmico.

En resumen, se reconoce la diversidad y validez de estas perspectivas didácticas para la resolución de problemas de programación computacional. Sin embargo, desde la intencionalidad de la presente investigación, centrada en la dinámica del proceso de enseñanza – aprendizaje de la algoritmización computacional, se considera la pertinencia del enfoque algorítmico a partir de considerar que las exigencias didácticas actuales deben estar encaminadas a desarrollar habilidades en el estudiante para comprender la situación problémica, diseñar una solución e implementarla en una computadora, lo que connota la naturaleza esencialmente algorítmica del proceso de resolución de problemas de programación computacional.

Los elementos explicados hasta aquí se consideran esenciales en la didáctica de la resolución de problemas de programación computacional, pues posibilitan potenciar la formación de habilidades algorítmicas en los estudiantes de las carreras de ciencias computacionales, lo que puede ser explicado a partir de la

correspondencia existente entre el procesamiento de la información que se concibe en la presente investigación y los elementos de la Teoría del Aprendizaje Significativo de D. P. Ausubel (1973, 1976, 2002).

Este aprendizaje significativo, a decir de su autor, es un proceso según el cual se relaciona un nuevo conocimiento o información con la estructura cognitiva del que aprende, de forma no arbitraria y sustantiva. Se produce en interacción con los aspectos relevantes presentes en la estructura cognitiva del sujeto aprendiz, los que reciben el nombre de subsumidores o ideas de anclaje (Ausubel, D. P., 1976, 2002).

La presencia de ideas, conceptos o proposiciones inclusivas y claras de Programación, disponibles en la mente del estudiante, es la que posibilita dotar de significado a la nueva información cuando interactúa con ella. Pero no se trata de una simple unión, sino que en este proceso la nueva información adquiere significado para el citado estudiante, produciéndose una transformación de los subsumidores de su estructura cognitiva, que resultan así progresivamente más diferenciados, elaborados y estables.

También constituye un referente psicológico el Enfoque del Procesamiento de la Información, a partir del cual se describen los fenómenos psicológicos como transformaciones de la información de entrada (input) a información de salida (output) (Gagné, E., 1991), facilitando la comprensión del proceso mental que desarrollan los estudiantes para resolver un problema de algoritmización computacional.

A propósito, una mirada dinámica y totalizadora al proceso de resolución de problemas de programación computacional se puede alcanzar teniendo como base los aportes hechos por H. C. Fuentes, E. C. Matos y S. S. Cruz (2004) desde la Teoría Holística Configuracional, al interpretar el movimiento de los diferentes procesos de enseñanza – aprendizaje a través de eslabones, comprendidos como expresiones de los complejos estadios por los que transita y que determinan su lógica interna. De manera que en la dinámica del procesamiento de la información que se lleva a cabo durante la resolución de un problema de programación, los estudiantes deben desarrollar un tránsito adecuado entre la comprensión de la situación problémica y la representación matemática y computacional del conocimiento extraído de la misma, enriqueciendo la fiabilidad de los resultados y su aprendizaje significativo.

Así, para tener éxito en la solución de una situación problémica será necesario facilitar la formación de un pensamiento algorítmico en el estudiante, que le permita relacionar los elementos de la citada situación con su conocimiento, para obtener una primera representación matemática de la misma, a partir de la cual lograr una segunda basada en estructuras lógico – computacionales.

La caracterización epistemológica realizada ha permitido observar que las diversas perspectivas con que se ha abordado el proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional, han producido importantes resultados, quedando sin explicar con suficiente profundidad las particularidades que distinguen la dinámica de la algoritmización que se da en este proceso, a través de argumentos que permitan alcanzar niveles superiores en la fundamentación de la representación matemática problematizada y su sistematización algorítmica, dinamizadora del proceso de resolución de dichos problemas, siendo ésta una de las causas por la que continúan manifestándose dificultades en su solución.

1.2 Caracterización histórica del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional

Para la construcción del marco teórico de la presente investigación se hace necesario conocer el comportamiento de determinadas regularidades del objeto y el campo de acción en el tiempo, consecuentemente el propósito del epígrafe es el establecimiento de las tendencias del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional y las especificidades de su dinámica de algoritmización. De manera que la evolución histórica de dicho proceso ha sido fundamentada a partir de asumir los siguientes indicadores:

- Desarrollo del hardware como premisa impulsora de la enseñanza – aprendizaje de la resolución de problemas de programación computacional.
- Evolución de los lenguajes de programación como condición para el desarrollo de la enseñanza – aprendizaje de la resolución de problemas de programación computacional.

- Principales enfoques didácticos del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional y su algoritmización.

El análisis tendencial que se expone se sustenta en la revisión de fuentes teóricas representativas de las principales características de la evolución histórica del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional. A partir de las mismas se realizó una periodización que quedó estructurada en dos etapas fundamentales:

- Primera etapa (1960 – 1983): Empleo de un enfoque algorítmico para la enseñanza – aprendizaje de la resolución de problemas de programación computacional.
- Segunda etapa (1983 – 2015): Predominio de un enfoque didáctico, centrado en los lenguajes de programación, para la enseñanza – aprendizaje de la resolución de problemas de programación computacional.

Antecedentes del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional

Los antecedentes de este proceso de enseñanza – aprendizaje no se pueden precisar sin hacer referencia al desarrollo de la computación como premisa del mismo. Es por ello que se hará una breve referencia al citado desarrollo, el que comienza en la segunda mitad del siglo XX con las primeras contribuciones de científicos de varios países. Tal es el caso del Dr. John. V. Atanasoff que en 1939 desarrolló la primera computadora digital electrónica (Grier, D. A., 2005), el Dr. Konrad Zuse que en 1941 terminó la primera computadora electro – mecánica completamente funcional, para la cual desarrolló un programa de control que hacía uso de los dígitos binarios (Grier, D. A., 2005) y Alan Turing que construyó en 1943 la primera computadora en utilizar tubos de vacío (Hodges, A., 1983) y científicos de la Universidad de Harvard y la International Bussines Machines (IBM) que crearon en 1944 la primera computadora construida a gran escala (Lee, J. A. N., 1995).

Posteriormente los doctores J. P. Eckert y J. W. Mauchly Jr. en 1946 desarrollaron una computadora electrónica completamente operacional a gran escala, la que se llamó Electronic Numerical Integrator And Computer (ENIAC). La ENIAC fue el primer ordenador totalmente electrónico, que ejecutaba 100,000 operaciones por segundo, a la que un equipo de seis mujeres matemáticas se ocupaban de programar, las que fueron inventando la programación a medida que la realizaban (Lee, J. A. N., 1995; Aspray, W., 1990). Estas científicas sentaron las bases para que la programación fuera sencilla y accesible a todos, crearon el primer set de rutinas, las primeras aplicaciones de software y las primeras clases de programación. Su trabajo modificó drásticamente la evolución de la programación entre las décadas del 40 y el 50 (Randall, A., 2006).

En 1949 J. V. Neumann, J. P. Eckert y J. W. Mauchly, construyeron la primera computadora que usaba el concepto de programa almacenado, Eletronic Discrete – Variable Automatic Computer, la que se constituyó en la primera con capacidad de almacenamiento de memoria e hizo desechar a los otros equipos que tenían que ser intercambiados o reconfigurados cada vez que se usaban. Esta fue la primera computadora electrónica digital de la historia, a partir de la cual fabricaron arquitecturas más completas (Aspray, W., 1990).

En 1951 aparece la serie UNIVAC construida por J. P. Eckert y J. W. Mauchly, la misma fue diseñada con propósitos de uso general pues ya podía procesar problemas alfanuméricos y de datos. Las tarjetas perforadas todavía conformaban el mayor recurso de alimentación de datos y toda la programación era muy compleja al realizarse en lenguaje de máquina o ensamblador, ésta fue la primera computadora comercial, que disponía de mil palabras de memoria central y podía leer cintas magnéticas (Norberg, A. L., 2005).

El desarrollo de la computadora trajo aparejado el desarrollo de los lenguajes de programación para dotar a estas máquinas de algoritmos o sentencias que le permitieran seguir un plan de trabajo de manera automática. Así en 1945 Konrad Zuse creó el Plankalkül, primer lenguaje de programación de la historia, el que nunca se llegó a implementar pero sirvió como aporte teórico (Zuse, K., 1993). En 1954 surgió el lenguaje Fortran, diseñado teniendo en cuenta que los programas serían escritos en tarjetas perforadas de 80

columnas. La inclusión en el lenguaje de un tipo de datos y de la aritmética de números complejos hizo al Fortran adecuado para aplicaciones técnicas tales como la ingeniería eléctrica (Metcalf y otros., 2004; Knuth, D. E., 1980). En 1958 surgió el Lisp que permitió que las funciones pudieran ser alteradas o creadas dentro de un programa sin un extensivo análisis sintáctico. En ese mismo año se publicó el lenguaje Algol, muy popular en las universidades en los años 60 y que influyó en varios lenguajes posteriores como Pascal, C. y ADA. En 1959 surgió el Cobol, con el objetivo de convertirse en un lenguaje de programación universal, orientado a la informática de gestión (Wirth, N. y Hoare, C. A. R., 1966; Backus, J. y otros., 1963).

Hasta ese momento el conocimiento de la programación era propio de los científicos dedicados al desarrollo de las computadoras, los que aprendían a programar a medida que iban construyendo o diseñando un lenguaje, pero el mismo avance tecnológico fue propiciando la necesidad de preparar profesionales que programaran estas computadoras; considerándose como el primer intento por enseñar computación las "Conferencias de la Escuela de Moore" que fue un curso sobre la construcción del ordenador digital electrónico sostenido en la universidad de Pensilvania en 1946 (Goldstine, H. H. y Goldstine, A.,1982).

Posteriormente, en 1947, se impartió en la Universidad de Harvard, Estados Unidos, la primera conferencia sobre el ordenador y se fundó la Association for Computing Machinery (ACM), sociedad profesional para organizar futuras conferencias sobre temas computacionales (Goldstine, H. H. y Goldstine, A., 1982).

Pero no es hasta 1949 que aparece el primer curso oficial de Ciencia de la Computación, brindado por la Universidad de Cambridge, Inglaterra y en 1953 la misma universidad inicia la primera carrera de Ciencia de la Computación a nivel mundial, con una duración de tres años (Denning, P. J., 2000).

Cabe señalar que en estos cursos solo se enseñaba a programar usando lenguaje máquina o ensamblador, el que es un lenguaje de ceros y unos con el que trabaja la máquina y que es realmente complejo, pues se debe escribir demasiado código para una simple sentencia, por lo que no era adecuado para aprender a programar.

En esta primera etapa las unidades de entrada para las computadoras utilizaban tarjetas perforadas. Al surgir

los lenguajes de alto nivel como el Fortran se perfeccionó la enseñanza de la Programación, pues se comenzaron a usar los diagramas de flujo para enseñar al estudiante a diseñar algoritmos y entender la secuencia de pasos u operaciones de los mismos. Hasta ese momento no se asociaba el término algoritmo con la programación, sino que su uso estaba generalizado al concepto matemático.

Es preciso aclarar que este desarrollo del hardware y el software no se produjo de la misma forma, ni llegó al mismo tiempo a todos los países. En el caso de Cuba, por ejemplo, la introducción de los primeros equipos para el procesamiento de datos se remonta a finales de los años veinte, para tratar el Censo de Población de 1930. Al inicio de la década siguiente, IBM instaló en La Habana una sucursal con rango de oficina central para el Caribe y América Central (Machado, M., 2000).

Primera etapa (1960 – 1983): Empleo de un enfoque algorítmico para la enseñanza – aprendizaje de la resolución de problemas de programación computacional

El acontecimiento que marca esta etapa en el ámbito educacional es la publicación, por parte de la Association for Computing Machinery (ACM), de los primeros diseños curriculares de los programas en ciencias de la computación, los que comenzaron a desarrollarse en 1960.

También en esta primera etapa continúan los avances tecnológicos, creciendo la capacidad de procesamiento de las computadoras y desarrollándose la programación de sistemas, además de crearse nuevos lenguajes. La generalidad de las computadoras de inicios de la etapa aún se programaba con cintas perforadas y otras por medio de cableado en un tablero. Los programas eran hechos a la medida por un equipo de expertos, los estudiantes no tenían contacto directo con las computadoras, se dedicaban a escribir instrucciones que luego los expertos procesaban para obtener los resultados y posteriormente se les daba participación en la verificación y corrección de los errores que aparecían.

Esta situación avanzó significativamente con la aparición de las computadoras personales, que tenían mejores circuitos, mayor memoria y unidades de disco flexible, acelerándose además con el surgimiento de programas de aplicación general, que podían ser empleados directamente por los estudiantes.

En resumen, se logra un gran avance tecnológico con la introducción de las computadoras con microprocesador. Aparecen los programas procesadores de palabras como el Word Star, la hoja de cálculo Visicalc y otros más que revolucionaron el rol de la computadora. El software empieza a tratar de alcanzar el paso del hardware y el estudiante se convierte en usuario de las computadoras.

Con respecto al software se inicia una verdadera carrera para encontrar la manera en que el estudiante, convertido en usuario, se capacitase con mayor facilidad y brevedad. Se pusieron a su alcance programas con menús (listas de opciones) que lo orientasen y otros programas que ofrecieran teclas de control y teclas de funciones para efectuar diversos efectos en el trabajo (Halvorson, M., 2008).

Se ofrecieron cursos para enseñar a usar los programas existentes, con el inconveniente de que ninguna solución para el uso de los programas era permanente, sino que cada nuevo programa requería aprender nuevos controles y nuevos menús. Esto motivó la necesidad de perfeccionar los equipos y programas de manera que se crease una relación amistosa entre el usuario y la computadora (Käasboll, J., 1998).

A inicios de esta etapa prevaleció el enfoque didáctico del manual o instruccionista, cuyo empleo fue necesario dado el desarrollo de la computación y su proceso de enseñanza – aprendizaje, carente de textos que se adecuaran a las necesidades escolares, con lo que los profesores tenían que acudir a los manuales técnicos (Esposito, C., 2001). Como consecuencia se priorizaron los elementos del lenguaje computacional, en detrimento de los procesos de búsqueda de resolución de problemas. Producto de esas condiciones de aprendizaje se veía frenada la independencia cognoscitiva de los estudiantes, que tenían que ceñirse a la aplicación de un conjunto de instrucciones predefinidas para realizar sus tareas docentes.

En 1962 aparece APL, lenguaje conciso, con una sintaxis muy sencilla, orientado al trabajo con matrices. A pesar de que el mismo era un lenguaje de alto nivel, los programas escritos eran difíciles de documentar y de comprender, por lo que nunca se usó con fines docentes (Selby, D., 2002; Alfonseca, M., 1974).

En 1964 surge el lenguaje Basic (Beginner's All – purpose Symbolic Instruction Code), el que originalmente fue desarrollado como una herramienta de enseñanza para facilitar la programación. Este lenguaje tuvo una gran aceptación y empleo en el proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional (Halvorson, M., 2008; Kemeny, J. G. y Kurtz, T. E., 1986).

En 1964 IBM propone el lenguaje PL1 para responder simultáneamente a las necesidades de las aplicaciones científicas y comerciales. Este lenguaje también fue ampliamente utilizado en la enseñanza de la Programación (Halvorson, M., 2008), pues facilitaba el desarrollo de habilidades de programación como la comprensión, interpretación, representación y generalización.

En 1967 es difundido oficialmente el lenguaje Simula que fue el primero de los lenguajes orientados a objetos. Este fue poco empleado en la docencia, pero que varios años después casi todos los lenguajes modernos comenzaron a utilizar sus principios de orientación a objetos, introduciéndose los conceptos de clases, objetos, instancias, herencia, polimorfismo, entre otros (Halvorson, M., 2008). En 1969 surgió el lenguaje B que no se pudo aplicar en las computadoras de la época, ni en la docencia, pero que posteriormente evolucionaría al lenguaje C (Johnson, S. C. y Kernighan B. W., 1983).

En 1970 aparece el lenguaje Pascal, que tenía como objetivo facilitar a los estudiantes el aprendizaje de la programación utilizando la estructuración de datos (programación estructurada). Con el tiempo su utilización excedió el ámbito académico para convertirse en una herramienta para la creación de aplicaciones de todo tipo. Este lenguaje facilitó el desarrollo de habilidades relativas a la estructuración computacional y la validación sintáctica y semántica, al contar con un compilador (Martinelli, O., 2006).

A finales de la década de 1970, el C empezó a sustituir al Basic como lenguaje predominante de programación de microcomputadoras (Ritchie, D. M., 1993). Cabe destacar que este lenguaje no se adoptó como lenguaje con propósitos docentes, sino que la gran mayoría de las universidades comenzó a usar el Pascal, empleando el enfoque algorítmico como enfoque didáctico fundamental para enseñar a programar, lo que sentó las bases para la formación de un pensamiento algorítmico de naturaleza computacional.

En el ámbito educacional, la ACM establece en el año 1968 el Currículum´68, que plasmó las principales características de la educación universitaria en computación y descripciones detalladas para programas académicos en Ciencias de la Computación, cursos y bibliografía. Contemplaba como cursos: Introducción a la Computación, Programación de sistemas y Sistemas de gran tamaño de procesado de la información.

Sin embargo, este currículo abarcaba la enseñanza de la computación de manera general, indicando el contenido que debía enseñarse, pero sin dar pautas sobre cómo enseñarlo. Esto en parte se debía a lo nueva que resultaba la introducción de la disciplina en los currículos de las carreras computacionales, no contando todavía con un diseño pedagógico bien concebido (Barchini, G., Sosa, M. y Herrera, S., 2004). No obstante, la enseñanza de la Programación y su inclusión en los currículos de varias carreras de ciencias, fue asumida de modo inmediato por muchas universidades del mundo.

Posteriormente, en el año 1978, la ACM presentó el Currículum´78, el cual tuvo un gran impacto sobre la enseñanza de las ciencias computacionales. En este se estableció el rol de la programación como eje central del proceso de enseñanza – aprendizaje de la computación, implantando un curso de Introducción a la Programación. Sin embargo, al igual que en el Currículum´68 dejaba claro qué enseñar, pero no cómo hacerlo (Zarazaga, F. J. y Alfonso, M. I., 2003; ACM´78).

Ahora bien, con el sistemático desarrollo del hardware y de los lenguajes de programación imperativos y estructurados, surgió la necesidad de perfeccionar los métodos y enfoques didácticos empleados hasta el momento para desarrollar el proceso de enseñanza – aprendizaje de la programación computacional

(Esposito, C., 2001). Es así que cobra auge el enfoque algorítmico, como la alternativa más efectiva para enseñar a programar. Este enfoque favoreció el desarrollo de habilidades de pensamiento lógico y abstracto, apoyándose en la representación de la situación problémica mediante pseudocódigos y diagramas de flujo; todo lo cual facilitó el aprendizaje de la programación (Cetín, I., 2013; Serna, E., 2011).

En el caso de Cuba, en este período se introdujeron computadores de la primera generación, adquiriéndose ya en 1965 las de la segunda generación, comenzándose en 1968 un plan de formación de profesionales de la computación, tanto dentro del país, como en el extranjero (Francia y URSS, fundamentalmente) (Blanco, L., 2003). En 1969, comenzó la importación de equipos, la organización de unidades operativas especializadas en técnicas informáticas y en aplicaciones, la producción nacional de equipos y el fortalecimiento de la formación de profesionales de la computación. Se hicieron los primeros ensayos de enseñanza de lenguajes de programación, introduciendo las primeras computadoras orientadas a la enseñanza media. En 1972, conjuntamente con la entrada de computadoras soviéticas, se comenzaron a estudiar los sistemas operativos DOS y OS, así como los lenguajes de programación Cobol, Fortran y PL1 (Blanco, L., 2003). Estos estudios se introdujeron fundamentalmente bajo el citado enfoque algorítmico en las carreras de Licenciatura en Matemática, Licenciatura en Ciencia de la Computación y en otras de ciencias naturales y técnicas.

En resumen, en esta primera etapa se produce un salto cualitativo en el desarrollo del hardware, que se constituye en premisa impulsora de la enseñanza – aprendizaje de la programación, dando lugar a que las computadoras pudieran estar al alcance de los estudiantes y la relación con las mismas se produjera de manera directa, sin mediación de los antiguos expertos. También evolucionan positivamente los lenguajes, desde la programación imperativa hacia la programación estructurada y posteriormente orientada a objetos.

Estas dos premisas dieron lugar a la aparición de los primeros diseños curriculares, al situar a la programación como eje central del proceso de enseñanza – aprendizaje de la computación. Consecuentemente se cambió la concepción de este proceso, cuyo enfoque didáctico transitó de lo

instruccionista a lo algorítmico, lo que representó un salto cualitativamente superior, al priorizarse la algoritmización con respecto a la enseñanza de la sintaxis de los lenguajes de programación.

Segunda etapa (1983 – 2015): Predominio de un enfoque didáctico, centrado en los lenguajes de programación, para la enseñanza – aprendizaje de la resolución de problemas de programación computacional

El inicio de esta segunda etapa está delimitado por importantes acontecimientos, destacándose la colaboración de la Computer Society of the Institute of Electrical and Electronics Enginereers (IEEE – CS), que en el año 1983 describió los tópicos más importantes en el área de las ciencias computacionales, utilizando un enfoque modular para organizar las materias y los cursos, lo que contribuyó a potenciar el proceso de enseñanza – aprendizaje de la programación computacional. También proliferaron los lenguajes de programación con fines docentes y continuó el proceso de transformación del hardware.

En esta etapa continuó el proceso de miniaturización de las computadoras, aumentando el poder de procesamiento de las mismas y su capacidad de almacenamiento. La generación de computadoras IBM y Pentium invadió el mercado internacional y apareció el Sistema Operativo Windows, que puso a la computación al alcance de cualquier persona que quisiera estudiarla. En lo referente a la educación se abrió un nuevo camino, comenzaron a aparecer los primeros software tutoriales y las primeras multimedias educativas. La computadora pasó, de objeto de estudio, a medio de enseñanza (Blanco, L., 2003).

Además predominó el desarrollo de los lenguajes de programación, entre los que apareció el ADA, en 1983; lenguaje multipropósito de programación estructurada y orientado a objetos. El lenguaje Pascal cobró su mayor auge hacia la segunda mitad de la década de los 80, con la creación del compilador Turbo Pascal para la IBM PC (Salazar, C. y Delrieux, C., 2004). En 1983 surgió el lenguaje C++, dando continuidad al C, pero ahora incluía la programación estructurada y la programación orientada a objetos. Este lenguaje sustituyó al Pascal en el proceso de enseñanza – aprendizaje de la programación, editándose libros especializados que

enseñaban a programar. Luego surgieron en 1985 Eiffel, 1987 Perl y en 1989 FL (Syme, D., Granicz, A. y Cisternino, A., 2010; Stroustrup, B., 2000).

Otro lenguaje de programación que apareció en la etapa fue el Haskell, en 1990, y en sus inicios se pensó con propósitos docentes, aunque no se materializó esta idea (Peyton, J. y Sheard, T., 2002). También se creó el Python en 1991 con fines académicos, poniéndose de moda en muchas universidades de Estados Unidos e Inglaterra (Knowlton, J., 2009). Aparecieron además en 1995 los lenguajes Java, Ruby y D, con la característica de ser todos orientados a objetos. El lenguaje Java se incluyó en muchos currículos de universidades de América Latina y Europa (Alexandrescu, A., 2010; Joyanes, L., 2004).

Así en la etapa, cada cambio de lenguaje acarreó consigo la inclusión de nuevos temas al contenido básico original de las asignaturas relacionadas con la programación, manifestándose un estrecho vínculo entre el lenguaje de programación y el paradigma de programación. Esto último se evidenció en el hecho de que mientras los lenguajes Fortran y Basic implantaron la programación imperativa y el lenguaje Pascal la programación estructurada, los lenguajes C++ y Java implantaron la programación orientada a objetos (Salazar, C. y Delrieux, C., 2004; Aho, A. V., Hopcroft, J. E. y Ullman, J. D., 1998). Lo anterior trajo como consecuencia didáctica el requerimiento de una mayor capacidad de abstracción para la resolución de problemas de programación computacional, la que fue insuficientemente trabajada en el proceso de enseñanza – aprendizaje, al desestimar la importancia formativa de la algoritmización como habilidad integradora que permite potenciar la comprensión, interpretación y modelación, previa a la implementación.

Consecuentemente, la introducción de los lenguajes orientados a objetos provocó que el énfasis en la formación computacional de los estudiantes se pusiera en el aprendizaje de los lenguajes y no en el desarrollo del pensamiento algorítmico, exigiéndoles capacidades de análisis y diseño de software, de manera prematura (Salazar, C. y Delrieux, C., 2004, Joyanes, L., 1998). Situación que se convirtió en tendencia

mayoritaria en las carreras de ciencias computacionales, en detrimento de la formación del pensamiento lógico – matemático, imprescindible en los profesionales de esta ciencia.

Por otro lado, en el año 1984, la Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura (UNESCO) desarrolló un Programa Modular de Informática para diferentes tipos de especialistas en la materia: programadores, analistas de sistemas e investigadores en informática, etc., lo que constituyó un paso de avance en la organización disciplinar.

En 1988, la ACM e IEEE desarrollaron el Currículum´88 y como resultado produjeron el informe "La Computación como disciplina" (Denning, P. J. y otros., 1989), dando un enfoque comprensivo y didáctico para la enseñanza de la Computación, centrado en la organización de tres procesos claves: teoría, abstracción y diseño. Lo más significativo de esta propuesta es que se planteó como uno de los objetivos del currículo, el desarrollo de habilidades para la resolución de problemas computacionales.

Ya a inicios de la década de los 90 la ACM y la IEEE elaboraron también el Curriculum´91, que difería de sus antecesores al ir a lo constitutivo y estar diseñado para soportar un desarrollo evolutivo e innovativo, dividido en varias áreas de conocimiento. Algunos de los temas recomendados fueron: “Conceptos fundamentales de la resolución de problemas”, “Proceso de desarrollo de software”, “Especificación y requisitos de software”, “Diseño e implementación de software” y “Verificación y validación”. Además planteaba que los programas de informática debían enseñar a los estudiantes cómo dominar al menos un lenguaje de programación. Recomendando que se les enseñara a ser competentes en lenguajes y a que hicieran uso de al menos dos paradigmas de programación (Cuevas, R. E., Miranda, A. y Martínez, J. M., 2011; ACM´91).

En el año 2001 la ACM e IEEE – CS elaboran la “Computing Currícula 2001 Task Force”, para obtener una perspectiva más holística y atender aspectos particulares de cada disciplina. Este nuevo currículum se orientó al dominio del conocimiento de la disciplina y a la didáctica de la computación, permitiendo crear una mayor especificación de qué y cómo se debía enseñar en ciencias de la computación. Este currículo 2001 contempló

la necesidad de conceptos y habilidades fundamentales en la práctica de la programación, con independencia del paradigma subyacente. Como resultado de esta propuesta, los principales temas considerados fueron: construcciones fundamentales de programación, algoritmos y resolución de problemas, estructuras de datos fundamentales, recursividad y programación controlada por eventos, dando relevancia a la algoritmización.

Así mismo, en el año 2005 la ACM e IEEE – CS elaboraron la "Computing Currícula 2005" donde delimitaron cinco disciplinas computacionales: Ciencia de la Computación, Ingeniería en Computadoras, Sistemas de Información, Tecnología de la Información e Ingeniería de Software. En este currículo se precisó el papel de la resolución de problemas de programación computacional y se reconoció que se podían usar dos estrategias en los cursos de programación: la primera consistente en enseñar de manera rápida los elementos del lenguaje, pasando directamente a programar, lo que pretendía aumentar la motivación en las asignaturas y la otra consistente en impartir cursos previos de algoritmización para crear ciertas bases del contenido y después comenzar con la programación en un lenguaje. Sin embargo, el énfasis en la enseñanza de la resolución de problemas de programación computacional se mantuvo centrado en los lenguajes de alto nivel.

Finalmente, en los años 2008 y 2013, la ACM e IEEE publican las "Computer Science Curricula 2008" y la "Computer Science Curricula 2013", en las que se mantuvieron las tendencias de enseñanza de la anterior currícula y se propusieron nuevas asignaturas enfocadas fundamentalmente al desarrollo de software.

Por otro lado, en la etapa predominaron los enfoques didácticos del modelo y del proyecto (Lissabet, A. y Cruz, M. A., 2011). El primero con el objetivo de provocar conflictos cognitivos que facilitaran la motivación de los estudiantes hacia la resolución de determinados problemas de programación especialmente escogidos para potenciar el aprendizaje. El segundo para favorecer la sistematización de los contenidos de programación a partir de su aplicación a la resolución de una situación problémica integradora, trabajada desde inicio de la asignatura en forma de proyecto. Con ambos enfoques se persigue alcanzar un nivel de

asimilación productivo, sin embargo no se refuerza su empleo sustentándolo mediante representaciones algorítmicas, previo a la implementación en un lenguaje específico.

También se empleó, pero en menor medida, el enfoque problémico para activar la enseñanza – aprendizaje de la programación, dirigiendo el proceso del pensamiento hacia la búsqueda de la solución de determinadas situaciones problémicas, explotando la contradicción que subyace en el proceso de adquisición del conocimiento (Neyret, R., 2005; Martínez, M., 1999). Sin embargo no fue suficientemente articulado dicho enfoque con el algorítmico, limitándose la efectividad en el desarrollo cognitivo del estudiante en cuanto a la resolución de problemas de programación computacional.

Estos enfoques desarrollados a nivel internacional repercutieron en la educación superior cubana, enfrascada en la introducción masiva de la Computación en todas las carreras universitarias (Machado, M., 2000; Mateu, M. M., 1998). A decir de E. Blanco (2003), la introducción de las microcomputadoras en el país significó un cambio de tecnología, tanto de hardware como software, así como de la filosofía de trabajo con estos equipos. Hubo que estudiar nuevos lenguajes de programación, asimilar sistemas de gestión de bases de datos, aprender la lógica de las hojas electrónicas, penetrar en la tecnología de los programas de dibujo y diseño, asimilar los procesadores de textos, aprender la mecánica del MS-DOS y sus comandos etc. Además se crearon carreras para formar especialistas en informática y tecnologías afines, como la Licenciatura en Cibernética Matemática y la Ingeniería en Sistemas Automatizados.

En resumen, en la etapa que se analiza prevaleció la tendencia de enseñar la programación de acuerdo con el lenguaje o paradigma del momento, no aprovechándose suficientemente las ventajas que brinda el enfoque algorítmico para la resolución de problemas de programación, lo que no ha facilitado el desarrollo de habilidades cognoscitivas básicas, relativas al pensamiento algorítmico, que les permitan desarrollarse como profesionales de las ciencias computacionales.

Finalmente, al analizar y sintetizar las etapas antes expuestas, se puede concluir que en la evolución histórica del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional se revelan como tendencias fundamentales los siguientes tránsitos:

- Desde un macro – hardware que requería la mediación de expertos, no facilitando una accesibilidad directa para el adecuado desarrollo del proceso de enseñanza – aprendizaje de la programación computacional, hacia un micro – hardware que significó un avance importante en dicho proceso al propiciar la interacción de los estudiantes con las computadoras para crear programas de aplicación general, posibilitando que la computadora se convirtiera en premisa impulsora del citado proceso.
- Desde el surgimiento y la supremacía de lenguajes de programación imperativos, hacia el uso de lenguajes de programación estructurada y orientada a objetos, los que requirieron mayor desarrollo de la capacidad de abstracción, al introducir conceptos como las estructuras de datos, las clases, la herencia, el polimorfismo y los objetos; conceptos que llevaron a centrar la atención de la enseñanza de la programación computacional en la sintaxis de los lenguajes, descuidando la formación de un pensamiento algorítmico, imprescindible en los profesionales de las ciencias computacionales.
- Desde el empleo de un enfoque didáctico algorítmico para desarrollar la dinámica del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional, hacia el predominio de enfoques que priorizan la enseñanza de la programación a partir del trabajo directo sobre un lenguaje, lo que ha propiciado que no se logre dotar al estudiante de los conocimientos esenciales para modelar las situaciones problémicas y verificar correctamente sus soluciones computacionales.

De todo lo anterior se deriva la necesidad de potenciar el desarrollo del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional desde una dinámica lógico – algorítmica que facilite la citada resolución, introduciendo nuevas relaciones que posibiliten al estudiante alcanzar una eficacia en la construcción del pensamiento algorítmico.

1.3 Caracterización del estado actual del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional en las carreras de ciencias computacionales de la Universidad de Oriente

La caracterización del estado actual del citado proceso se sustentó en dos medios de diagnóstico: una encuesta a estudiantes de segundo año de las carreras de ciencias computacionales de la Universidad de Oriente y una entrevista a profesores de la asignatura de Programación para dichas carreras.

Para la encuesta, la población estuvo conformada por 182 estudiantes de segundo año de las cuatro carreras de ciencias computacionales de la Universidad de Oriente, a inicios del curso 2012 – 2013 y estructurada por: 31 estudiantes de Ingeniería Informática, 77 de Ingeniería en Telecomunicaciones y Electrónica, 24 de Licenciatura en Ciencia de la Computación y 50 de Ingeniería en Automática. No obstante, atendiendo a la necesidad de economizar recursos en la investigación, no se extrajo información directamente de cada unidad experimental, sino que fue necesario seleccionar una muestra aleatoria de la población bajo estudio.

Cabe señalar que se esperaba que los resultados de la encuesta fuesen favorables respecto a habilidades básicas que están presentes en el proceso de resolución de problemas de programación computacional, dado que la población con la que se trabajó estuvo conformada por estudiantes que ya habían cursado y aprobado la asignatura de Programación en su primer año.

Para la determinación del tipo de muestreo a emplear se tuvo en cuenta la homogeneidad existente entre los programas de la asignatura de Programación para las cuatro carreras del citado centro, cuyos contenidos coinciden en más del 80%. Además, se tuvieron en cuenta los resultados generales del diagnóstico inicial. Consecuentemente, se consideró cada carrera como un conglomerado contentivo de las características de las variables a estudiar, pudiéndose concluir la pertinencia del uso de un muestreo probabilístico por conglomerados bietápico (Cochran, W. G. ,1980; Yamane, T., 1980).

La primera etapa de este muestreo consistió en la selección aleatoria de dos conglomerados para conformar la muestra, siendo elegidas las carreras de Licenciatura en Ciencia de la Computación e Ingeniería Informática. En la segunda etapa se pasó a la determinación de una muestra probabilística a partir de un muestreo aleatorio simple en cada uno de los conglomerados seleccionados, resultando conformada por 18 estudiantes de Licenciatura en Ciencia de la Computación (75 % de los 24) y 23 de Ingeniería Informática (74,19 % de los 31), para un total de 41 estudiantes a ser encuestados.

La encuesta a los 41 estudiantes tuvo por objetivo profundizar en el objeto y el campo de acción definidos, para lo cual se empleó la variable operativa «aprendizaje de la algoritmización para la resolución de problemas de programación computacional», que se define como la apropiación y aplicación de contenidos de algoritmización a la resolución eficaz y eficiente de problemas de programación computacional. Esta variable fue operacionalizada sobre la base de los 16 indicadores o ítems siguientes:

1. Replanteo los problemas a resolver con mis propias palabras.
2. No compruebo la solución de cada problema.
3. Leo el planteamiento del problema varias veces antes de tratar de resolverlo.
4. No utilizo gráficos, tablas, ecuaciones y funciones para representar el problema.
5. Distingo la información relevante de la irrelevante en cada problema antes de solucionarlo.
6. No comienzo a resolver el problema hasta estar seguro de que he interpretado de manera clara y precisa cada uno de sus elementos y tengo una visión integrada de ellos.
7. Exploro diferentes estructuras computacionales (for, if, then, while, etc.) antes de diseñar un algoritmo de solución para el problema.
8. Para dar solución a un problema diseño el programa sin tener en cuenta un orden lógico para utilizar las estructuras computacionales (for, if, then, while, etc.).
9. Estimo la respuesta final después de concebir una vía para alcanzar la solución.

10. Después de hallar la solución no pruebo otros datos para ejecutar (correr) el programa.
11. Reflexiono sobre el método o los métodos que utilizo para solucionar un problema, después de solucionado.
12. No diseño un algoritmo antes de implementar en un lenguaje particular.
13. Utilizo pseudocódigo para diseñar el algoritmo antes de implementarlo.
14. Al resolver un problema comienzo a implementarlo en un lenguaje sin utilizar pseudocódigos.
15. Dominar la sintaxis o reglas de un lenguaje es más importante que saber algoritmizar.
16. Si un programa cumple con la sintaxis de un lenguaje y se ejecuta correctamente, entonces el resultado esperado cumple con el objetivo para el que fue implementado.

Para la elaboración de los ítems se utilizaron afirmaciones y negaciones, con el propósito de reducir el sesgo en la opinión de los estudiantes, el que se introduce al utilizar un solo tipo de estas (Bayona, J. A. y otros, 2005) (Ver Anexo No. 4). Luego, con el objetivo de compatibilizar las respuestas, se trabajó con la transformación de las negaciones que se presentaron en los ítems pares (**I-2**, **I-4**, **I-6**, **I-8**, **I-10**, **I-12**, **I-14** e **I-16**). Así todos los ítems resultantes estuvieron elaborados de forma afirmativa para facilitar el análisis de los datos. Las respuestas se clasificaron según cada uno de los 16 ítems anteriormente declarados, teniendo en cuenta una escala de Likert (ordinal), con cinco niveles de respuesta.

Consecuentemente, para el procesamiento de la información de la citada encuesta se estructuraron clases sobre la base de la media y los umbrales de clasificación (Gorina, A. y Alonso, I., 2012) (Ver Anexo No. 5), quedando conformadas las cuatro clases naturales (muy favorable, favorable, desfavorable y muy desfavorable) que se muestran en el propio anexo.

Una síntesis descriptiva de los resultados obtenidos para los dos conglomerados muestreados (Licenciatura en Ciencia de la Computación e Ingeniería Informática) se presenta en las Tablas 6.1 y 6.2 del Anexo 6, las que evidencian que los mismos en general son favorables, como bien se esperaba para este tipo de

estudiantes, que habían vencido la asignatura de Programación. Sin embargo, un análisis más detallado de la información muestra que existen dificultades relacionadas con los ítems (**I-6**, **I-12**, **I-13**, **I-14**, **I-15** y **I-16**) que responden directamente a la algoritmización en el proceso bajo estudio.

Ahora bien, el interés fundamental que persiguió la investigación, con respecto al tipo de información que se demandaba de la encuesta aplicada a los estudiantes de segundo año de las carreras de ciencias computacionales, era obtener interpretaciones sobre la base de las inferencias estadísticas realizadas a la población objeto de estudio (las cuatro carreras de ciencias computacionales) con relación a la variable operativa, sustentada en la información de la muestra seleccionada.

Consecuentemente, la Tabla 6.3 del Anexo 6 muestra, para cada operacionalización de los indicadores y para la variable operativa bajo estudio, la estimación a nivel poblacional de la media y la varianza. Además presenta el error absoluto, el coeficiente de variación y el intervalo de confianza del 95% de confiabilidad. En la última columna de la misma se brinda la valoración realizada sobre la variable operativa estudiada y sobre su respectiva operacionalización, tomando como base la información elaborada a partir del análisis de los datos. Una síntesis de los resultados obtenidos en dicha tabla se muestra en la Figura 1.1.

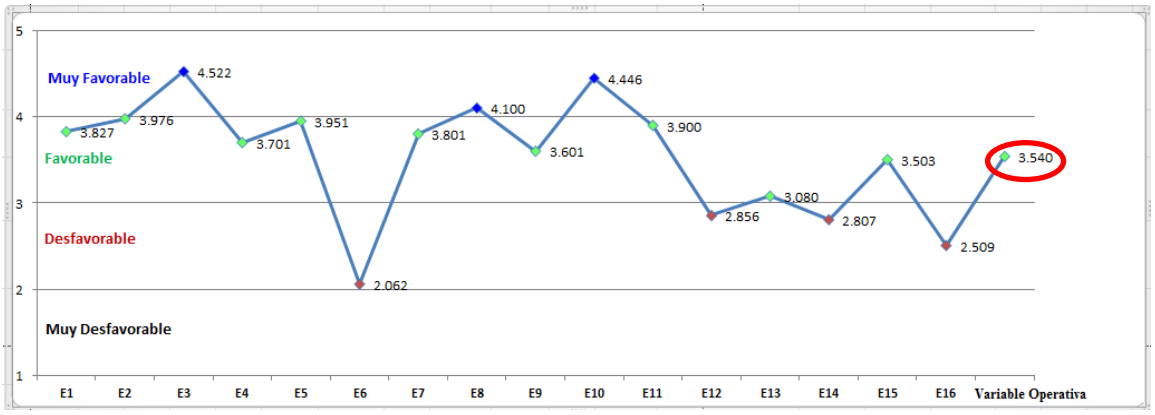


Figura 1.1. Representación gráfica de la clasificación en cuatro clases de las inferencias estadísticas obtenidas para los 16 ítems de la encuesta aplicada a los estudiantes de las cuatro carreras de ciencias computacionales, así como el valor de la inferencia para la variable operativa «aprendizaje de la algoritmización para la resolución de problemas de programación computacional»

Se concluye que el resultado de la variable operativa inferido a la población es **favorable** y que las insuficiencias se localizan en los ítems **I-6**, **I-12**, **I-14** e **I-16**, relacionados fundamentalmente con el proceso de algoritmización computacional. En esta dirección, a continuación se describen los resultados de los ítems señalados:

En el ítem **I-6** la media alcanzada es de 2,062, lo que hace que sea clasificado como **desfavorable** con rasgos latentes de **muy desfavorable**. Estos resultados indican que existen dificultades al "momento de comprender e interpretar una situación problémica", pues los estudiantes comienzan a resolver una situación problémica sin estar totalmente claros de lo que se les exige y esto conlleva en muchas ocasiones a resolver un problema distinto al que se plantea. Esta dificultad, si bien ha sido tratada didácticamente desde diversas ciencias, requiere de un tratamiento particular desde la Didáctica de la Programación Computacional, pues se debe formar en el estudiante la necesidad de comprender e interpretar una situación problémica antes de proceder a su resolución matemático – computacional.

Los ítems **I-12** e **I-14,** correspondientes a "diseñar el algoritmo antes de implementarlo en un lenguaje" y a "la resolución de un problema de programación computacional directamente en un lenguaje sin usar pseudocódigos", obtienen valores 2,856 y 2,807 de la estimación para la media poblacional, clasificados como **desfavorable**. Estos valores revelan que los estudiantes en su mayoría no crean el algoritmo ni usan pseudocódigos antes de implementar la solución en un lenguaje de programación, lo que da cuenta de la insuficiente valoración que hacen de la importancia que tiene llevar a cabo el diseño del algoritmo mediante pseudocódigos para tener un desempeño exitoso en la resolución de problemas de programación.

En el ítem **I-16** se obtuvo una media de 2,509, clasificada como **desfavorable**, relativo a "cuando un programa cumple con la sintaxis de un lenguaje y su ejecución es correcta, cumple con el objetivo para el que fue implementado". Esto refleja que la mayoría de los estudiantes centra su atención en el dominio de la parte sintáctica del lenguaje computacional, en detrimento de los aspectos cualitativos o semánticos que son

inherentes a todo problema de programación, es decir, una vez que los estudiantes verifican que el programa se ejecuta correctamente (corre), asumen que cumple con la intencionalidad deseada, sin realizar una detallada validación semántica de las condiciones cualitativas de la situación problémica de partida.

Hasta aquí se ha tenido en cuenta, para la interpretación de los resultados, el carácter bipolar de la escala (positiva, negativa), analizando únicamente el polo negativo. Sin embargo, resulta provechoso profundizar en los niveles cualitativos en los que se estructura el polo positivo, con la intención de discernir aquellos aspectos en los que se puede perfeccionar el proceso bajo estudio.

En consecuencia, los ítems **I-1**, **I-2**, **I-5**, **I-4**, **I-7**, **I-9**, **I-11**, **I-13** y **I-15** fueron evaluados de **favorable** y los tres primeros con rasgos de **muy favorable.** De lo anterior se deriva la necesidad de seguir perfeccionando los procesos relacionados con la comprensión de la situación problémica inicial, la representación matemático – computacional de dicha situación (en particular la representación algorítmica mediante pseudocódigos) y la validación sintáctico – semántica final para alcanzar la solución. Este resultado guarda correspondencia con el hecho de que en el diagnóstico inicial de la presente investigación se evidenció la existencia de bajos niveles en la calidad de la promoción en la asignatura de Programación, pues en general los encuestados son estudiantes que han vencido los contenidos de dicha asignatura, pero no con un alto aprovechamiento académico (calidad de las calificaciones).

Ahora bien, a partir de la inferencia estadística realizada, los ítems (**I-3**, **I-8** e **I-10**) alcanzaron valoraciones de **muy favorable**, los cuales hacen referencia a la "lectura reiterada de la situación problémica, al orden lógico para utilizar las estructuras computacionales y a la prueba de las soluciones con varios juegos de datos". Esto se corresponde con una de las acciones que más automatizan los estudiantes dentro del proceso de resolución de problemas de programación computacional. En general los mismos pasan, en reiteradas ocasiones del citado proceso, de la lectura del problema a su implementación computacional, sin realizar

una comprensión totalizadora de la situación problémica inicial, por lo que realmente sí desarrollan la lectura, pero no necesariamente llegan a una adecuada comprensión.

Asimismo, se aprenden las estructuras básicas de programación y su orden lógico, pero frecuentemente, ante una situación problémica concreta, no manifiestan habilidades para la identificación, selección y jerarquización de dichas estructuras, resultándoles más fácil automatizar el funcionamiento de las mismas, que aplicarlas a situaciones problémicas. Esto es debido a que las primeras habilidades son netamente reproductivas, mientras que las segundas son productivas, requiriendo el concurso de un pensamiento algorítmico.

Resulta significativo el hecho de que estas insuficiencias permanezcan en estudiantes de segundo año, que ya han cursado y aprobado la asignatura de Programación y aún subvaloran las potencialidades de la algoritmización para estructurar el proceso de resolución de problemas de programación computacional. Sin embargo, esto corrobora la sistematización realizada en los epígrafes anteriores del presente capítulo, en los que se reveló un abandono del enfoque didáctico algorítmico para desarrollar la dinámica del mencionado proceso, priorizando aquellos enfoques centrados en el trabajo directo con un lenguaje de alto nivel.

No obstante, si se desea tener cierto nivel de completitud de la información referida al comportamiento fáctico del objeto de investigación y el campo de acción, sería muy arriesgado guiarse solamente por los resultados aportados por la encuesta. De aquí que se considerase conveniente realizar una entrevista a los profesores que imparten la asignatura de Programación en las carreras de ciencias computacionales.

Consecuentemente, para la entrevista se eligió el 80% de profesores de las cuatro carreras de ciencias computacionales de la Universidad de Oriente, teniendo en cuenta: años de experiencia, categoría docente y grado científico. La misma tuvo por objetivo profundizar en el objeto y el campo de acción de la investigación, para lo cual se tomaron tres indicadores y se realizó el procesamiento de la información obtenida, detallando las opiniones que con mayor frecuencia fueron emitidas por los profesores entrevistados (ver anexos 7 y 8).

Sintetizando los aspectos detallados en los citados anexos se llega a que en el indicador “momento del proceso de programación en el que se confrontan las mayores dificultades”, las respuestas más relevantes de los profesores se pueden clasificar según dos insuficiencias relativas a: “la aplicación de un lenguaje de alto nivel para la programación sin concebir y emplear un algoritmo previo” y “las habilidades para algoritmizar”.

Respecto a la primera insuficiencia, coinciden en que el uso de un lenguaje de alto nivel para enseñar de manera directa a programar no es adecuado, pues el estudiante pierde demasiado tiempo aprendiendo la sintaxis y no dedica tiempo a crear habilidades de algoritmización, que es lo que le permitirá resolver un problema correctamente. También precisan que cuando se tiene el algoritmo diseñado, su traducción a un lenguaje específico es sencilla y como en la actualidad se cuenta con compiladores que señalan cualquier error sintáctico, el problema se reduce a que el algoritmo diseñado cumpla con la intencionalidad deseada.

Además reconocen que no es en el momento de ejecución y validación del programa, en el que los estudiantes confrontan las mayores dificultades, sino en el proceso de diseñar un programa para dar solución a un problema determinado, añadiendo que el diseño del algoritmo es lo más importante.

En el segundo caso, relativo a las insuficiencias en las habilidades para algoritmizar, los profesores reconocen la algoritmización como parte esencial del proceso de programación, enmarcada en la etapa de elaboración del programa, como una estructura que tiene carácter generalizador, al permitir su implementación en cualquier lenguaje computacional. También hacen énfasis en la etapa previa de análisis e interpretación de la situación problémica, evidenciando la pertinencia de diseñar un algoritmo que refleje la solución del problema que se está intentando resolver, antes de pasar a su implementación.

Por otro lado, al preguntarles sobre otros aspectos relevantes para el proceso de enseñanza – aprendizaje de la Programación, los profesores consideran el uso de paquetes matemáticos, así como de otros productos computacionales que permitan que los estudiantes se representen el funcionamiento de la computadora

cuando se diseña un algoritmo. También valoran como relevante la formación de un pensamiento abstracto en función de la programación y de habilidades para optimizar los algoritmos.

Finalmente, al indagar respecto a "¿cómo le gustaría a usted aprender programación?", contestaron en tres direcciones sumamente importantes, una referida al aumento del trabajo de la formación algorítmica (ya sea usando pseudocódigos, diagramas de flujo o software para algoritmizar), otra dirigida hacia el reforzamiento de la formación matemática (en enseñanzas precedentes y en la enseñanza superior) y la última encaminada a comenzar la enseñanza de la programación desde enseñanzas precedentes.

Una vez aplicados los dos instrumentos se procedió a realizar una triangulación analítica para corroborar y contrastar las insuficiencias en el proceso de resolución de problemas de programación computacional. Esto se hizo sobre la base de la información obtenida de la aplicación de la encuesta a estudiantes y la entrevista a profesores de Programación. Es así que para la triangulación sólo se utilizaron los indicadores valorados como **desfavorable**, para encausar la investigación hacia el perfeccionamiento del mencionado proceso.

Para el caso del ítem **I-6**, se pudo apreciar una correspondencia entre las valoraciones de los estudiantes y las opiniones de los profesores, en ambos casos se concluye que existen insuficiencias en el momento de comprender e interpretar una situación problémica. Ya desde la caracterización epistemológica de la presente investigación estos resultados eran previsibles, en tanto la etapa de "comprensión" en muchos casos es subestimada por los estudiantes, motivo por el cual resultan comunes los errores durante el proceso de resolución de una situación problémica. En esta dirección son numerosos los didactas que han arribado a similares conclusiones partiendo de observaciones hechas al desempeño de los alumnos (Guibert, N., Guittet, L. y Girard, P., 2005b; González, W., Estrada, V. y Martínez, M., 2006; Pérez, R., 2009).

En el caso de los ítems **I-12** e **I-14**, las valoraciones realizadas por los estudiantes dan cuenta de una insuficiente apreciación de la importancia que tiene realizar el diseño del algoritmo mediante pseudocódigos, en aras de tener un desempeño exitoso en la resolución de problemas de programación computacional.

Sin embargo, la mayoría de los profesores coinciden en la importancia de recuperar la enseñanza de la programación computacional mediante un enfoque algorítmico que utilice pseudocódigos o diagramas de flujos, atendiendo a que esto permite centrarse en la formación del pensamiento algorítmico del estudiante novel sin perder demasiado tiempo aprendiendo la sintaxis de los lenguajes de alto nivel. Estos resultados corroboran las inconsistencias epistémicas develadas en el campo de acción de la presente investigación, así como la necesidad de introducir nuevas propuestas teórico – metodológicas que posibiliten paliarlas.

Finalmente, en el ítem **I-16** las valoraciones de los estudiantes apuntan a un mayor dominio de la parte sintáctica del lenguaje computacional que a los aspectos semánticos de la algoritmización, lo que reduce la eficiencia en la resolución de problemas de programación computacional. No obstante, la mayoría de los profesores precisan que lo más importante es lograr concebir el algoritmo, pues su traducción a un lenguaje específico no es compleja y puede hacerse con ayuda de un software; pero las prácticas del proceso de enseñanza – aprendizaje de la programación computacional muestran una realidad muy distinta a esta visión profesoral, connotándose la enseñanza del lenguaje de programación de alto nivel, en oposición a las vías de estimulación para la formación de un pensamiento algorítmico.

Todo lo analizado en este epígrafe permitió evidenciar la necesidad de transformar el proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional y la manera en que se lleva a cabo la formación del pensamiento algorítmico, para lo cual es pertinente elaborar propuestas teórico – metodológicas que revelen las esencialidades del citado proceso, haciendo énfasis en la algoritmización.

Conclusiones

1. A partir de las inconsistencias teóricas y prácticas precisadas en la caracterización epistemológica del objeto y el campo de la investigación, se manifiesta la necesidad de revelar las especificidades que distinguen la dinámica de la algoritmización en la resolución de problemas de programación

computacional, a través de una lógica integradora y coherente, que posibilite alcanzar niveles de interpretación superiores de la esencia de la situación problémica que se aborda.

2. Las tendencias históricas precisadas permitieron revelar un insuficiente empleo del enfoque didáctico algorítmico para desarrollar la dinámica del proceso de enseñanza – aprendizaje del proceso de resolución de problemas de programación computacional, así como el predominio de enfoques que priorizan la enseñanza de la programación desde el trabajo directo sobre un lenguaje, lo que da cuenta de la necesidad de introducir nuevas relaciones que posibiliten la formación de un pensamiento algorítmico.
3. El diagnóstico del estado actual del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional en las carreras de ciencias computacionales de la Universidad de Oriente, permitió constatar insuficiencias que tienen su base en la dinámica de la algoritmización que se debe llevar a cabo para la resolución de los problemas de programación computacional, de aquí la necesidad de elaborar propuestas que expliquen dicho proceso de algoritmización.
4. La caracterización que desde el punto de vista epistemológico e histórico se hace del objeto de estudio y el campo de acción de la presente investigación, así como de su estado actual en las carreras de ciencias computacionales de la Universidad de Oriente, encaminan la misma hacia la elaboración de un modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional, que permita superar las insuficiencias teóricas y metodológicas existentes en el proceso de enseñanza – aprendizaje de la algoritmización computacional, desde una lógica que integre la representación matemática problematizada y su sistematización algorítmica.

CAPÍTULO II: CONSTRUCCIÓN TEÓRICO – PRÁCTICA DE LA DINÁMICA DEL PROCESO DE ALGORITMIZACIÓN EN LA RESOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN COMPUTACIONAL

Introducción

En el capítulo se fundamenta un modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional, el que posibilita una interpretación esencialmente superior del proceso de algoritmización, dando lugar a que emerja una lógica integradora entre la representación matemática de la situación problémica y su generalización mediante pseudocódigos. Dicho modelo se concreta en un sistema de procedimientos didácticos que persigue orientar a los profesores de Programación para que conduzcan adecuadamente la citada dinámica.

2.1 Fundamentos teóricos del modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional

El proceso de algoritmización computacional demanda una reconstrucción que tenga en cuenta la didáctica de la resolución de los problemas de programación computacional, en su integración con la Matemática y otras ciencias que faciliten explicar pertinentemente la dinámica de su lógica algorítmica. En consecuencia, desde lo **epistemológico**, la modelación se sustenta en el sistema de categorías de la Teoría Holístico – Configuracional adelantada por H. C. Fuentes, E. C. Matos y S. S. Cruz (2004). El haber asumido dicho sistema categorial propició una base adecuada a la naturaleza dinámica del modelo y a su carácter totalizador. Así mismo, facilitó revelar su regularidad, con lo que se profundizó en

el conocimiento del comportamiento de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional.

Desde lo **psicológico** se asumió, de la Teoría del Aprendizaje Significativo de D. P. Ausubel (1973, 1976, 2002), la forma en que se incorporan los nuevos conocimientos a la estructura cognitiva del sujeto, lo que se logra cuando el mismo relaciona estos nuevos conocimientos con los anteriormente adquiridos. De aquí que, durante el proceso de resolución de problemas de programación sea posible estimular la generación de un pensamiento algorítmico en el estudiante, que le permita relacionar la representación matemática de la situación problémica con su conocimiento sobre las estructuras lógico – computacionales, para generalizar dicha representación expresándola mediante pseudocódigos.

También constituye un referente psicológico el Enfoque del Procesamiento de la Información, particularmente sus resultados sobre el conocimiento humano, en términos de los procesos por medio de los cuales las entradas sensoriales son transformadas, reducidas, almacenadas, recuperadas y usadas (Gagné, E., 1991); lo que permite comprender cómo el estudiante percibe los elementos que componen las situaciones problémicas y las relaciones que se dan entre estos, los transforma y reduce a representaciones matemáticas, para luego generalizarlos mediante pseudocódigos, almacenándolos y recuperándolos para ser usados cada vez que requiera su reanálisis a lo largo del proceso de algoritmización computacional. Así, la representación matemática y la generalización computacional de una situación problémica puede verse como el procesamiento de la información que ésta brinda, regulada por la base de conocimientos y experiencias del estudiante que la aborda.

Desde la **didáctica de la resolución de problemas matemáticos** se asumió la categoría de representación de problemas matemáticos propuesta por I. Alonso (2001). Esto permitió comprender que en el proceso de resolución de un problema de programación computacional el estudiante debe partir del

análisis de la situación problémica que aborda e ir realizando interpretaciones y representaciones (matemáticas y computacionales) cada vez más precisas, que lo acerquen a la solución algorítmica.

Desde la **didáctica de la resolución de problemas de programación computacional**, se asumió de los investigadores N. Arellano y otros (2014), J. J. Arellano y otros (2009, 2012) y N. Guilbert, L. Guittet y P. Girard (2005a, 2005b, 2006) la concepción de que el aprendizaje de la Programación no puede reducirse al aprendizaje de la sintaxis de un lenguaje, debiendo priorizarse la enseñanza de la algoritmización a partir del uso de pseudocódigos o diagramas de flujo. Esto sirvió de sustento a toda la modelación realizada, dado que la hipótesis que se sigue en la presente investigación se basa en el hecho de que para enseñar programación es necesario enseñar primero a algoritmizar como habilidad básica que permitirá formar el pensamiento algorítmico y que la enseñanza debe centrarse en fomentar en el estudiante la necesidad de construir algoritmos en pseudocódigos.

2.2 Modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional

La dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional es comprendida como todas aquellas relaciones, regularidades y metodologías que se dan en la didáctica del proceso de algoritmización computacional, que permiten establecer y predecir el movimiento del mismo, desde una lógica integradora de los conocimientos matemáticos y computacionales, para potenciar la formación de un pensamiento algorítmico computacional.

El modelo que se presenta de la citada dinámica está conformado por cuatro dimensiones, las que son expresión de sus movimientos internos y permiten revelar la transformación del mencionado proceso. Dichas dimensiones son: la construcción lógico – matemática, la orientación matemático – algorítmica, la estructuración algorítmico – generalizadora y la validación algorítmico – computacional (Salgado, A., Gorina, A. y Alonso, I., 2013).

Dimensión de la construcción lógico – matemática

La explicación de esta dimensión lleva a explicitar su movimiento a partir de las relaciones esenciales que se producen entre sus configuraciones. Dicho movimiento se revela en la representación matemática de la situación problémica, al ser la misma síntesis de la contradicción que se establece entre la comprensión de la situación problémica y la interpretación matemática de dicha situación.

Así, la configuración de ***comprensión de la situación problémica***, que da inicio al movimiento de la dimensión, es expresión de los sucesivos acercamientos mentales que debe realizar el estudiante para comprender el significado de una determinada situación problémica, a los efectos de su solución desde una lógica algorítmica. En esta dirección se debe tener en cuenta que hasta que éste no comprenda bien la situación que aborda, todo el trabajo que realice para resolverla puede resultar improductivo.

La comprensión implica un análisis de la situación problémica, es decir, su fragmentación en partes para examinar detalladamente cada una de estas, con el propósito de identificar los objetos reales, matemáticos o computacionales que las conforman, sus características, funciones y relaciones. Todo esto conducirá al estudiante a una primera comprensión de los componentes de la situación problémica, la que deberá ir perfeccionando a lo largo de todo el proceso de algoritmización.

Ahora bien, la comprensión de una situación problémica depende de los conocimientos que posea el estudiante, no sólo computacionales y matemáticos, sino también sobre el mundo que le rodea, pues tendrá que ir imaginando objetos y relaciones, en correspondencia con los objetos y relaciones externas que presenta la citada situación y tratar de exteriorizar dichos objetos ideales mediante la expresión oral, símbolos escritos, dibujos o esquemas. De aquí que los conocimientos y experiencias sobre el mundo sean sumamente importantes para que pueda lograr una adecuada comprensión de la situación.

La presencia de conocimientos lingüísticos, por ejemplo, es muy necesaria, ya que el estudiante debe comprender el lenguaje y las experiencias por medio de las cuales recibe el planteamiento de la

situación. Debe explicarse que no basta con el conocimiento del lenguaje en el que se expresa la situación, debe comprenderse el contexto en que se inscriben los hechos que la conforman, es decir; el conocimiento semántico y el conocimiento de cómo funciona el mundo en el que se da la misma.

Si todos estos conocimientos resultan suficientes para procesar la información que brinda la situación problémica, es muy probable que el estudiante pueda lograr una adecuada comprensión de la misma, la que facilitará su posterior solución. Sin embargo, esta comprensión no es suficiente si no se complementa con una **interpretación matemática de la situación problémica,** como su par dialéctico, expresión del proceso de reconstrucción matemática que realiza el estudiante, a partir de establecer asociaciones y relaciones entre los objetos que intervienen en la situación problémica, con lo que puede atribuir un significado a dicho proceso en función de la problemática real que pretende resolver desde un conocimiento matemático, adquiriendo con ello un sentido diferente y cualitativamente superior.

Aquí resulta de vital importancia el proceso encaminado a identificar las condiciones y exigencias de la situación problémica (ambas conformadas por los objetos, características y relaciones que ya han sido analizadas por él previamente), a partir de lo cual deberá establecer asociaciones y relaciones que le permitan aislar los objetos, características y relaciones que son esenciales, desechando toda la información que no se considera relevante a los efectos de la exigencia de la situación.

Ahora bien, si se considera el conocimiento matemático como un conjunto de esquemas mentales en los que el estudiante ha organizado los conceptos, principios, fórmulas y procedimientos de esta ciencia, se podrá entender que a través de esos esquemas es que este puede relacionar y organizar la información nueva, para interpretarla en forma significativa, en aras de seleccionar la relevante.

Como parte de este proceso interpretativo el profesor deberá cuidar que el estudiante identifique y se apropie del nexo y la diferencia existentes entre el símbolo, que desempeñará el rol de representante, y el objeto

representado, ya que es el símbolo el medio a través del cual materializa la abstracción, y a la vez el medio material sobre el que trabaja el pensamiento abstracto.

Entre las dos configuraciones anteriormente explicadas se establece una relación dialéctica, que se manifiesta entre lo general y lo particular, a partir de concebir que en la medida en que el estudiante va comprendiendo la situación problémica desde una imagen general y totalizadora, se favorece una adecuada interpretación de la misma desde su particularidad matemática, y en este mismo proceso de interpretación matemática, a su vez, se verifica su pertinencia y su adecuación o no a las exigencias de dicha situación, generando una nueva comprensión, más profunda, de la misma. La contradicción se manifiesta porque un cambio en la comprensión de la situación significa una transformación en su interpretación matemática y una profundización en esta última puede enriquecer la comprensión de la situación, negando la que se tenía inicialmente.

Ahora bien, la relación que se establece entre la comprensión de la situación problémica y su interpretación matemática se sintetiza dialécticamente en la configuración **representación matemática de la situación problémica,** la que es concebida como la concreción, en un modelo matemático, de los resultados de la comprensión e interpretación que ha realizado el estudiante de la situación problémica, que puede exteriorizar mediante la expresión oral, símbolos matemáticos, fórmulas, gráficos, tablas, etc.; de manera que se facilite la visualización y conceptualización matemática de los objetos y relaciones inherentes a la situación problémica, analizada desde la lógica esencial de esta ciencia.

Es así que la representación matemática de la situación problémica se caracteriza por tener como contenido un conocimiento generalizado de dicha situación, de aquí que tenga como condición una determinada abstracción, que considera la situación en su esencia, ya que una vez que el estudiante, mediante los procesos de comprensión e interpretación ha aislado los objetos, características y relaciones que son esenciales, los integra en un nuevo objeto, más simplificado y significativo que la situación original.

Además, esta representación concebida por el estudiante puede pasar por diferentes grados de generalización, lo que hace que no se considere como una reproducción mecánica e invariable de la percepción de la situación, sino que se asuma como una configuración dinámica. La relación entre la representación y la situación problémica regulará las modificaciones a las que será sometida la primera.

Por otro lado, la representación matemática de la situación problémica permitirá la determinación y activación del conocimiento a emplear en su algoritmización y solución. De aquí que al comienzo del proceso los conocimientos del estudiante le permitirán obtener una representación matemática de la situación y, en consecuencia, serán recuperados de su memoria los procedimientos concretos de algoritmización, si los conoce. En tal sentido, lo que guía la recuperación de los procedimientos apropiados es la representación matemática que de la situación se forma el estudiante, por ello se considera crucial dicha representación para tener éxito en la algoritmización, al ser la que determina qué conocimiento computacional se activará en la memoria.

Un aspecto que el profesor no puede perder de vista es que cada individuo crea sus representaciones matemáticas, de ahí que diferentes estudiantes pueden crear diferentes representaciones de una misma situación problémica. Esto más que una dificultad puede favorecer el proceso de enseñanza – aprendizaje de la algoritmización, siempre que sea bien aprovechado.

Otro aspecto que puede ser explotado es el uso de varias representaciones matemáticas de un mismo problema, lo que permitirá hacer comparaciones y evaluar cada una de ellas, de manera que se logre disponer de un mayor número de herramientas matemáticas para resolverlo, pudiendo valorar y escoger las menos complejas, lo que influirá en la identificación y selección de las estructuras computacionales que se deberán asociar a los objetos matemáticos durante el proceso de algoritmización computacional.

Todo lo visto hasta aquí consolida, en un primer nivel de sistematización matemática, la manera en que el estudiante se representa y construye su modelo mental, resultado de la abstracción realizada, como

punto de partida para nuevas representaciones, en una búsqueda y resignificación constante de las relaciones lógico – matemáticas modeladas.

Las relaciones entre estas tres configuraciones anteriores permiten connotar la **dimensión de la construcción lógico – matemática** como expresión del movimiento que se establece entre la compresión de la situación problémica y su interpretación matemática, que se sintetiza en la representación matemática de la situación problémica, como un primer estadio de desarrollo en la dinámica de algoritmización computacional que se propone.

Esta dimensión es expresión del carácter de integración y sistematización lógico – matemática, desplegado por el estudiante para desarrollar sus potencialidades en aras de comprender e interpretar pertinentemente, desde sus conocimientos matemáticos, el significado de los objetos y relaciones contenidos en la situación problémica, lo que permite alcanzar estadios superiores en la representación matemática realizada durante el proceso resolutor (ver figura 2.1).

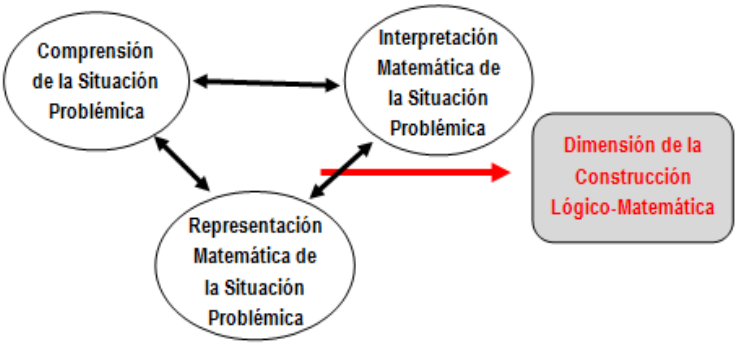


Figura 2.1: Dimensión de la construcción lógico – matemática.

A su vez la configuración síntesis, **representación matemática de la situación problémica**, es expresión de otro movimiento del proceso, que a través de la dimensión de la orientación matemático – algorítmica expresa la relación que se establece entre la identificación de estructuras lógico – computacionales y la integración jerárquica de dichas estructuras.

Dimensión de la orientación matemático – algorítmica

La configuración **identificación de estructuras lógico – computacionales** es interpretada como aquel proceso mediante el cual el estudiante reconoce las estructuras algorítmicas necesarias para garantizar

una lógica coherente en el proceso de algoritmización. Este reconocimiento se logra a partir del análisis de la representación matemática de la situación problémica, contentiva de los rasgos esenciales de dicha situación, en correspondencia con los conocimientos computacionales y matemáticos que posee.

Así, cuando el estudiante ya tiene concebida una representación matemática de la situación problémica, tendrá que distinguir dentro de un conjunto de estructuras algorítmicas conocidas por él, aquellas que le serán útiles para transformar los objetos y relaciones que aparecen en la citada representación, a partir del conocimiento de las características y funciones de dichas estructuras.

Ahora bien, la identificación de estructuras lógico – computacionales presupone el dominio de ciertas habilidades y operaciones. No se puede identificar una estructura si no se ha aprendido a determinar las propiedades y funciones de la misma por medio de la comparación con otras estructuras, esto es lo que le permitirá diferenciarlas. Pero determinar las propiedades y funciones de una estructura es insuficiente, deben saberse diferenciar las propiedades y funciones esenciales de las no esenciales, como habilidad clave para lograr la identificación. Esto requiere del dominio del concepto de propiedad y función, además de habilidades para diferenciar en las estructuras diversas propiedades y características.

Sin embargo, esta identificación de estructuras lógico – computacionales por sí sola no garantiza que el estudiante conciba un algoritmo que conduzca a la solución de la situación problémica que aborda y para lograrlo el profesor deberá evidenciar la conveniencia de llevarla a cabo en estrecha relación con la **integración jerárquica de estructuras lógico – computacionales**, como configuración que garantiza la selección, análisis y concatenación adecuada de las estructuras lógicas, previamente identificadas, para conformar el algoritmo que constituirá la estructura funcional del proceso de programación.

De esta forma el estudiante estará en condiciones de distinguir y ordenar dichas estructuras, a partir de una comparación previa, constituyéndose en expresión concreta de una elección lógico – computacional

– estructurada, que deberá asegurar la eficiencia algorítmica del proceso y la solución de la situación problémica objeto de análisis.

Todo lo anterior implica el desarrollo de una dinámica formativa que facilite al estudiante la identificación e integración jerárquica de las citadas estructuras para dar significado a las secuencias lógicas desde una interpretación algorítmico – computacional, lo que se constituye en una vía para dirigir las operaciones y procedimientos individuales, a partir de un proceso de ordenación, disposición y estructuración algorítmica. Esto propicia una pertinencia de la intencionalidad lógico – computacional que lo llevará a garantizar la eficiencia algorítmica del proceso.

Aquí se debe garantizar que la integración de las estructuras computacionales concebidas por el estudiante llegue a ser óptima, mediante el empleo de herramientas de control, iterativas o de definición en el orden exacto. Esto permitirá que, cuando se implemente el algoritmo que se está diseñando, se haga un uso eficiente de la memoria del equipo de cómputo y que el tiempo de ejecución sea mínimo.

La optimización realizada contribuye al desarrollo de habilidades de programación, en lo referente al diseño y la concepción de algoritmos. Así, una vez culminada esta fase, aunque no se pueda asegurar que el estudiante haya logrado crear un pseudocódigo bien establecido, sí se debe haber producido una apropiación de todos los elementos necesarios para construirlo. De aquí que en esta etapa se connote el carácter integrador y desarrollador que posee la algoritmización como proceso computacional, capaz de propiciar el desarrollo de capacidades cognitivas al tener que integrar y optimizar simultáneamente.

La interrelación entre estas configuraciones conforma un par dialéctico, que deviene en unidad indisoluble, ya que en la lógica de algoritmización computacional la identificación de estructuras lógico – computacionales lograda por el estudiante se constituye en base de toda integración jerárquica de la misma, al determinar su selección, ordenamiento y jerarquización lógica, dándole un sentido de estructura algorítmica a dicha integración jerárquica; a su vez, esta integración contiene su propia

identificación de estructuras. La contradicción se manifiesta porque la identificación de una nueva estructura lógico – computacional conlleva a una jerarquización diferente y la aparición de una jerarquización ineficaz implica una revaloración de la pertinencia de la identificación realizada.

El proceder algorítmico del estudiante se connota entonces en el propio proceso de programación computacional, desde una doble perspectiva: por un lado, permitiéndole planificar y evaluar su acciones y estructuras lógicas en la resolución de la situación problémica; y por otro, retroalimentando constantemente su intencionalidad algorítmica a partir del reconocimiento, selección y concatenación jerárquica de las estructuras algorítmicas, para obtener niveles cada vez más esenciales y precisos de la representación matemática inicial. De aquí que esta representación sirva de guía para dirigir y operacionalizar la dinámica que se construye, convirtiéndose en un proceso que permite el autocontrol del estudiante sobre sus acciones matemático – algorítmicas.

De esta forma la representación matemática de la situación problémica se resignifica desde la relación entre la identificación e integración jerárquica de estructuras lógico – computacionales, ya que esta relación aporta al estudiante nueva información que le permite valorar dicha representación a partir de una perspectiva algorítmico – computacional, elevándola a un nivel algorítmico cualitativamente superior.

Es así que surge un nuevo movimiento de la dinámica bajo estudio, a partir de las relaciones que se establecen entre las tres configuraciones citadas, lo que deviene en un segundo nivel de sistematización matemática que está dado por la **dimensión de la orientación matemático – algorítmica** (ver figura 2.2).

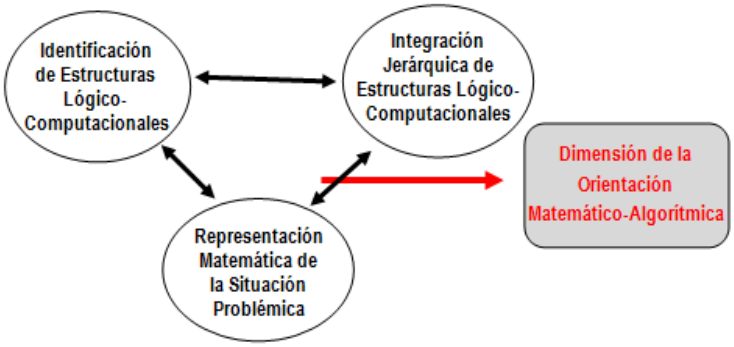


Figura 2.2: Dimensión de la orientación matemático-algorítmica.

Esta dimensión es interpretada como expresión del proceso que direcciona la construcción de una estructura algorítmico – funcional para la resolución de la situación problémica, desde su representación matemática. Su esencia está en la precisión de las vías y procedimientos lógicos que posee el estudiante, inmerso en un sistemático análisis de los objetos y relaciones matemáticas, así como de las estructuras algorítmico – computacionales pertinentes.

Dimensión de la Estructuración Algorítmico – generalizadora

Como resultado de la relación dialéctica que se establece entre la identificación e integración jerárquica de estructuras lógico – computacionales se configura, con un carácter de integralidad cualitativamente superior, la configuración de **generalización pseudocodificada de la representación matemática**, como constructo teórico que es síntesis y dinamizador de la lógica de algoritmización computacional desplegada. Esta es expresión del proceso de resignificación y reestructuración de la representación matemática, creada a partir de una identificación e integración sistemática de estructuras computacionales, lo que conduce a una representación más general, expresada mediante pseudocódigos. Este proceso de generalización trasciende el simple aprendizaje de las características de un lenguaje específico, encaminándose hacia el reconocimiento y aplicación del pseudocódigo, como herramienta base para la escritura del algoritmo.

En síntesis, al sistematizar e integrar, lógica y coherentemente, la relación dialéctica que se establece entre las configuraciones identificación de estructuras lógico – computacionales e integración jerárquica de dichas estructuras, se potencia el surgimiento de la generalización pseudocodificada de la representación matemática, como cualidad

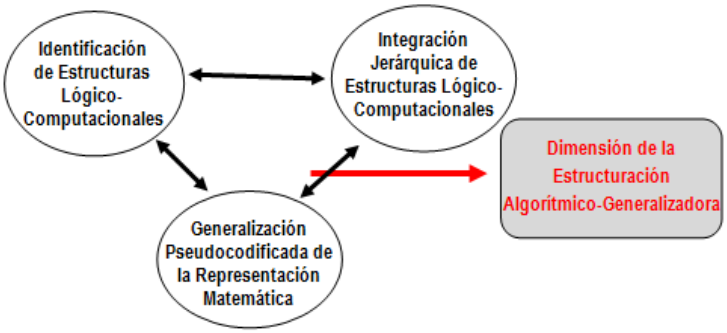


Figura 2.3: Dimensión de la estructuración algorítmico – generalizadora.

que da cuenta del carácter generalizador que debe tener todo algoritmo, dando lugar así a la **dimensión de la estructuración algorítmica generalizadora** (ver figura 2.3).

Esta estructuración se constituye en un primer nivel de sistematización computacional, que aproxima a la vía de solución del problema, lo que permite una explicitación de la representación matemática realizada a través de la generalización como habilidad fundamental, a partir del uso de pseudocódigos a modo de herramienta principal para realizar la traducción del lenguaje matemático al computacional.

Dimensión de la validación algorítmico – computacional

La configuración síntesis **generalización pseudocodificada de la representación matemática**, es expresión, a su vez, de un nuevo movimiento interno en la dinámica del proceso de algoritmización computacional, el que se expresa a través de la relación que se establece entre la valoración sintáctica de la representación computacional y la valoración semántica de la representación computacional.

La configuración **valoración sintáctica de la representación computacional** es interpretada como los sucesivos refinamientos que debe realizar el estudiante antes, durante y después de estructurado el algoritmo, a partir de tomar en cuenta la sintaxis del pseudocódigo, a los efectos de regular y evaluar dicho proceso, lo que favorece la toma de decisiones para el perfeccionamiento constante del mismo.

Una premisa esencial a considerar por el docente es que el empleo de pseudocódigos no significa la ausencia de errores, ya que pueden crearse algoritmos en pseudocódigo que no sean correctos debido a errores sintácticos. Esto connota el valor didáctico que tiene la formación de habilidades para optimizar la valoración sintáctica de los algoritmos, máxime cuando se reconoce que dos estudiantes pueden crear algoritmos diferentes para dar respuesta a un mismo problema, obteniendo resultados equivalentes.

En este momento es importante que el profesor considere que el lenguaje algorítmico, usando pseudocódigos, facilita la representación de las construcciones básicas de los lenguajes de programación,

con la ventaja de conservar cierta coincidencia con el lenguaje natural y con algunos convenios sintácticos inherentes a estos lenguajes, tales como asignaciones, ciclos y condicionales.

Por otro lado, el docente debe hacer notar a sus estudiantes que se podría utilizar para describir un algoritmo el lenguaje natural pero, debido a los innumerables problemas que introduce, como la imprecisión o la ambigüedad, es más conveniente utilizar otras herramientas algorítmicas como el pseudocódigo, que describan con mayor exactitud la secuencia de acciones y el orden en el que han de ejecutarse.

A pesar de su flexibilidad, el pseudocódigo tiene que atenerse a una serie de normas para que los algoritmos construidos resulten legibles, claros y fácilmente codificables, por lo que el profesor debe verificar que los identificadores usados tengan un significado de acuerdo con su contenido y que el conjunto de sentencias sea completo; es decir, que permita especificar cualquier tarea a realizar con suficiente detalle. También deberá comprobar que contenga un conjunto de palabras reservadas, quedando bien reflejado el flujo de control del algoritmo, que es el orden temporal en el cual se deben ejecutar los pasos individuales del mismo.

Sin embargo, la valoración sintáctica de la representación computacional no resulta categoría suficiente para lograr una generalización pseudocodificada sino es en relación con la **valoración semántica de la representación computacional**, como proceso que permite garantizar la pertinencia de la representación realizada por éste a partir de pseudocódigos; lo que se logra mediante la precisión de su efectividad, en correspondencia con la intencionalidad deseada para dar respuesta a la situación problémica.

En este estadio del proceso el estudiante debe verificar la validez de la solución computacional que propone, lo que le permite ratificar si la estructuración del pseudocódigo es correcta en términos del contenido. Este análisis semántico debe estar alejado de cualquier herramienta formal, desde el punto de vista computacional, pues se sustenta principalmente en la verificación de que el algoritmo cumple con su función generalizadora a partir de análisis progresivos de corridas no automatizadas y cada vez más abarcadoras.

Es así que, una vez que el estudiante ha escrito el algoritmo utilizando una herramienta adecuada como el pseudocódigo, se hace necesario verificar que este realiza las tareas para las que fue diseñado y que produce los resultados correctos y esperados a partir de la información de entrada. Es importante precisar que este proceso se conoce como prueba del algoritmo y consiste básicamente en recorrer todos los caminos posibles del mismo, comprobando en cada caso que se obtienen los resultados esperados; para lo cual será conveniente realizar una ejecución manual del algoritmo, con datos significativos que abarquen todo el posible rango de valores, así como evidenciar que en cada caso la salida coincide con la esperada.

Así el docente debe enfatizar que la aparición de errores puede conducir a tener que rediseñar determinadas partes del algoritmo que no han funcionado correctamente y a aplicar de nuevo el proceso de localización de errores, definiendo nuevos datos de prueba y recorriendo de nuevo el algoritmo con estos. Por consiguiente, se debe potenciar la formación del estudiante en cuanto a la necesaria corroboración de la pertinencia de la representación computacional, a partir del fundamento de que, aún cuando se haya realizado correctamente la modelación matemática y computacional del problema, resulta imprescindible optimizar el pseudocódigo propuesto, con el objetivo de ganar en eficiencia y eficacia de los resultados, lo que redundará en la apropiación de una cultura de perfeccionamiento de los algoritmos computacionales.

Estas dos últimas configuraciones devienen en unidad dialéctica, ya que la valoración sintáctica permite al estudiante una evaluación constante de las estructuras computacionales, garantizando así la valoración semántica de la representación computacional realizada por este. De manera que un error desde el punto de vista sintáctico puede conducirlo a una falsa interpretación semántica, por lo que, en la misma medida en que se compruebe la intencionalidad y pertinencia del algoritmo, se ratifica y transforma su significado desde la coherencia algorítmica alcanzada. A su vez esta transformación y corroboración de la validez del contenido computacional obtenido conduce inevitablemente a la necesidad de evaluar y modificar continuamente la escritura sintáctica del pseudocódigo, con lo que se transforma la valoración semántica anterior.

Las relaciones que se establecen entre las categorías generalización pseudocodificada de la representación matemática, valoración sintáctica de la representación computacional y valoración semántica de la representación computacional, permiten revelar un nuevo movimiento en la lógica algorítmica modelada, dando lugar a la **dimensión de la validación algorítmico – computacional** (ver figura 2.4). Dicho movimiento se constituye en un segundo nivel de sistematización computacional, expresando la verificación didáctica de la validez y pertinencia de las estructuras computacionales empleadas, a partir de connotar la exactitud en la valoración y generalización de la lógica construida desde la pertinencia y certeza del algoritmo realizado.

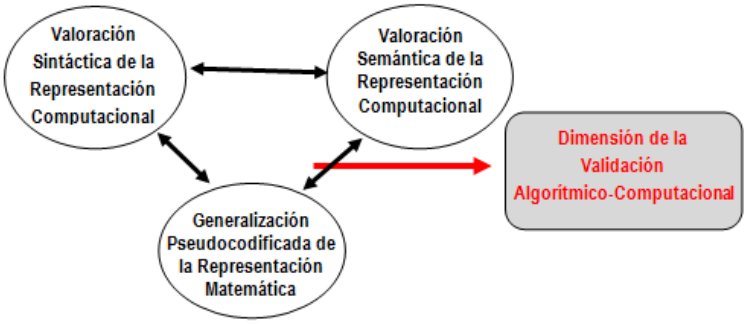


Figura 2.4: Dimensión de la validación algorítmico – computacional.

Es así que esta dimensión garantiza la corrección, tanto sintáctica como semántica, de la solución propuesta, permitiendo al estudiante diseñar y crear algoritmos computacionales que resuelvan una determinada situación problémica de manera eficiente y eficaz, acorde con los requerimientos de la misma, lo que deviene cualidad generalizadora que condiciona la confirmación de la funcionalidad y viabilidad algorítmica de la lógica didáctica desplegada en el proceso de programación computacional.

Ahora bien, entre las configuraciones síntesis **representación matemática de la situación problémica** y **generalización pseudocodificada de la representación matemática** existe una relación dialéctica que se manifiesta en la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional. Esta relación se fundamenta al tener en cuenta que la representación matemática de la situación problémica, en sus dos niveles, posibilita una construcción lógico – matemática y una orientación matemático – algorítmica de la solución, que como cualidades emergentes dan cuenta de la regulación y perfeccionamiento de la citada representación, en función de su problematización computacional. Esto sienta

las bases para que a partir de la generalización pseudocodificada de la representación matemática emerjan otros dos niveles de sistematización, el de estructuración algorítmico – generalizadora y el de validación algorítmico – computacional, los que garantizan que se haya realizado una adecuada generalización algorítmica a partir de pseudocódigos.

De lo anterior se deduce que la unidad de la citada relación dialéctica se manifiesta en el hecho de que no es posible realizar una generalización pseudocodificada de la representación matemática si no es a partir de la representación matemática de la situación problémica, ya que esta última se constituye en condición de partida para lograr la primera. Pero una representación matemática por sí sola no garantiza el éxito en la solución algorítmica, porque si bien proporciona una aproximación a la misma, no asegura la especificidad computacional requerida para alcanzarla.

A su vez, la contradicción se expresa en que un cambio en la representación matemática de la situación problémica genera una transformación de la generalización pseudocodificada de la representación matemática, exigiendo una adecuación a las nuevas condiciones y, por otro lado, la aparición de inexactitudes, imprecisiones y falta de pertinencia del algoritmo propicia que surjan inconsistencias y contradicciones que niegan la representación matemática de la situación problémica realizada y exige que esta última profundice en el estudio de los objetos y relaciones que representan a la situación problémica.

Esta contradicción entre las configuraciones síntesis es superada mediante la formación de un **pensamiento algorítmico computacional**, el que a su vez es expresión de la relación que se establece entre las cuatro dimensiones del modelo y que funciona de manera integral y se desarrolla en el proceso de resolución de las situaciones problémicas, en el que los procesos de representación y generalización se constituyen en síntesis sucesivas de interpretación de dichas situaciones, dando lugar a nuevas síntesis reflexivas, cualitativamente superiores. Este pensamiento es entendido como una forma de razonamiento lógico e integrador, centrado en la algoritmización, que actúa en el proceso de resolución de problemas de programación computacional,

facilitando el tránsito entre los procesos de representación matemática y generalización pseudocodificada de dichos problemas y posibilitando al estudiante la obtención de soluciones computacionales eficientes y eficaces (ver figura 2.5).

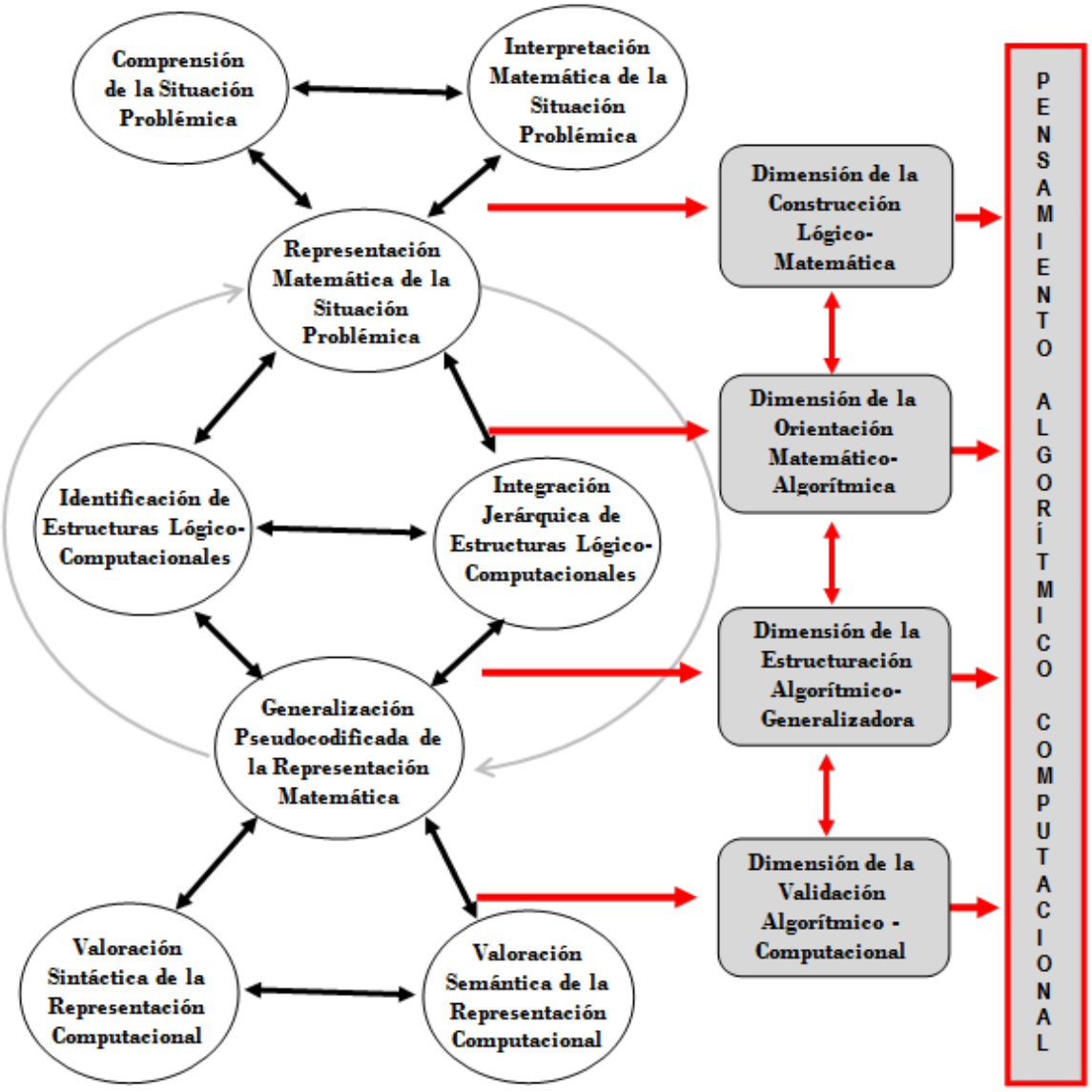


Figura 2.5: Modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional.

Estos movimientos dan lugar a sucesivas transformaciones en los sujetos implicados en la resolución de las situaciones problémicas, pues durante el desarrollo de la explicada dinámica éstos van alcanzando niveles más esenciales y profundos de síntesis reflexivas, los que les facilitarán la algoritmización a partir de una

apropiación de patrones que aporta la Matemática para el análisis y representación de las situaciones problémicas, así como de herramientas computacionales para la estructuración y validación de los algoritmos.

Ahora bien, la formación en el estudiante de un **pensamiento algorítmico computacional**, debe crear las condiciones para el reforzarmiento de sus habilidades de abstracción y decodificación de situaciones problémicas, con el objetivo de modelarlas matemática y computacionalmente. También se potencian destrezas para la identificación y empleo de estructura lógico – computacionales, su estructuración y validación usando pseudocódigos. Consecuentemente el propósito final del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional debe estar encaminado a la formación de este tipo de pensamiento.

Así, de la modelación de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional emerge un **sistema de relaciones esenciales**, que permite interpretar su comportamiento y transformación. Dicho sistema está integrado por las siguientes relaciones:

- La representación matemática de la situación problémica como síntesis de la relación entre la comprensión de la situación problémica y su interpretación matemática, da lugar a un movimiento que genera una construcción lógico – matemática.
- La orientación matemático – algorítmica como expresión de una síntesis representativa de la situación problémica, se origina por la contradicción que emerge entre la identificación y la integración jerárquica de estructuras lógico – computacionales.
- La generalización pseudocodificada de la representación matemática, como síntesis de la relación entre la identificación de estructuras lógico – computacionales y la integración jerárquica de estas, es generadora del movimiento que conduce a una estructuración algorítmico – generalizadora.

- El movimiento que da lugar a la validación algorítmico – computacional se origina de una generalización pseudocodificada de la representación matemática, que es síntesis de la relación entre la valoración sintáctica de la representación computacional y la valoración semántica de dicha representación.

A partir del estudio de estas relaciones se devela como **regularidad**, la lógica integradora entre la representación matemática de la situación problémica y la generalización pseudocodificada de la representación matemática, como condición imprescindible para el desarrollo de un pensamiento algorítmico computacional, relación que a la vez da cuenta de la doble modelación, matemática y computacional, que debe experimentar una situación problémica en el camino algorítmico que conduce a su solución.

Ahora bien, para llevar a la práctica el sistema de relaciones esenciales y la regularidad de la modelación realizada, se requerirá de la elaboración de un **sistema de procedimientos didácticos** que oriente a los docentes en la conducción de la formación del **pensamiento algorítmico computacional**, en aras de elevar la eficacia y eficiencia resolutora de las situaciones problémicas, con lo que se contribuirá al perfeccionamiento de su formación profesional desde la enseñanza – aprendizaje de la algoritmización.

2.3 Sistema de procedimientos didácticos para la algoritmización computacional

El sistema de procedimientos didácticos para la algoritmización computacional está conformado por un conjunto de acciones, lógicamente estructuradas y ordenadas, que facilitan el desarrollo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional. Se ha construido a partir del método Sistémico Estructural Funcional, por la necesidad de instrumentar la citada dinámica mediante un conjunto de procedimientos integrados. Es por tanto un instrumento de intervención didáctica, que tiene como **objetivo** la orientación intencional a los profesores de Programación para la conducción del proceso de enseñanza – aprendizaje de la algoritmización.

En cuanto a su **alcance**, cabe precisar que el mismo está diseñado para potenciar la formación de aquellos estudiantes que se inician en el estudio de la Programación, a la que debe anteceder el aprendizaje de

contenidos de algoritmización computacional, los que pueden formar parte de dicha asignatura o constituirse en cursos introductorios (independientes de todo lenguaje y paradigma computacional) para las carreras de ciencias computacionales. Así mismo, está dirigido a estudiantes de aprovechamiento docente promedio, quedando a decisión de los profesores hacer los énfasis necesarios en la formación de aquellos de bajo y alto aprovechamiento docente (Salgado, A., Alonso, I. y Gorina, A., 2014b).

La intervención didáctica que se pretende llevar a cabo mediante este sistema de procedimientos favorecerá el desarrollo de acercamientos cada vez más profundos y esenciales a la algoritmización computacional, como elemento dinamizador del proceso de resolución de problemas de programación, con lo que se estará contribuyendo al desarrollo de un pensamiento algorítmico computacional en el estudiante, independiente del lenguaje de programación, en virtud del cual se deberán ir obteniendo niveles cualitativamente superiores de formación computacional en dichos estudiantes. Es decir, que la lógica del sistema de procedimientos promueve transformaciones cada vez más relevantes, que contribuyen al perfeccionamiento del proceso de enseñanza – aprendizaje de la algoritmización en la programación computacional, las cuales se convertirán en una guía para el logro de una autonomía resolutora, a partir de un trabajo más consciente y estable, que viabilice el auto – desarrollo formativo de los mismos, de aquí su **carácter didáctico.**

Otras cualidades que lo distinguen desde su condición de sistema, son las siguientes:

- Es un **sistema abierto**, dando cuenta de ello el hecho de que está sometido a múltiples influencias externas y la dinámica que promueve permite su remodelación y mejoría constante. Esto posibilita su continuo perfeccionamiento a partir de las nuevas tecnologías computacionales y el desarrollo de la propia didáctica de la programación.
- Es **recursivo** ya que adquiere sentido de los procedimientos que lo conforman (sus partes) y dichos procedimientos adquieren significado a través de su integración sistémica (todo), lo que da cuenta de su coherencia.

- Manifiesta su **sinergia** en el pensamiento algorítmico computacional que se configura en la dinámica de la resolución de problemas de programación, como nueva cualidad totalizadora alcanzada en su implementación.
- Su **entropía** puede ser provocada por las insuficiencias que presentan los estudiantes para concebir un algoritmo coherente a la hora de resolver mediante la programación una situación problémica, así como por la resistencia de los actores del proceso para aceptar los cambios que en el mismo implica la introducción de la nueva dinámica de algoritmización, la que exige de una mayor preparación matemática y computacional, que requiere de una profundización en el trabajo con representaciones que involucran objetos, relaciones matemáticas y pseudocódigos. Otra fuente de entropía es la asociada al proceso de comunicación, el que no siempre logra la eficiencia necesaria.
- La **homeostasis** es favorecida cuando el profesor adquiere conciencia de la necesidad de realizar una doble modelación (matemática y computacional) para la resolución de los problemas de programación computacional y es capaz de formar habilidades en sus estudiantes para realizar las acciones previstas en el sistema de procedimientos didácticos. Además se puede potenciar valiéndose de las posibilidades que ofrecen los programas analíticos de Programación, de las Prácticas Laborales y de los Trabajos de Curso, en función de adiestrar a los estudiantes en la dinámica lógico – algorítmica que se propone. También se puede fomentar dicho equilibrio mediante el empleo de software educativos, que permitan profundizar en la interpretación de la algoritmización de diversas situaciones problémicas.
- Su **autodesarrollo** se expresa en el carácter flexible, abierto, dinámico que posee, el que facilita al profesor su sistemática adecuación a determinadas circunstancias contextuales, que permiten su progresivo perfeccionamiento y desarrollo.

- Las principales **premisas** para la implementación del sistema de procedimientos didácticos son:
  a) Plan de Estudio con objetivos en correspondencia con la resolución de problemas de programación computacional, que permitan realizar ajustes para llevar a cabo la enseñanza de la algoritmización.
  b) Buena preparación del claustro en cuanto a conocimientos matemáticos, computacionales (principalmente técnicas y métodos de algoritmización) y la lógica matemática.
  c) Buena preparación metodológica para asimilar el modelo y el sistema de procedimientos propuestos.
  d) Experiencia en la enseñanza de la resolución de problemas de programación computacional.
  e) Indicaciones de la correspondiente dirección docente – metodológica, estableciendo la necesidad de desarrollar estrategias específicas para avanzar en el campo de la resolución de problemas de programación computacional desde la algoritmización.
  f) Apropiada motivación de estudiantes y profesores por el trabajo que desarrollan para la formación de habilidades de resolución de problemas de programación computacional.
  g) Dominio básico, por parte de los estudiantes, de los conocimientos y habilidades precedentes, referidos a la resolución de problemas matemáticos.
  h) Existencia de recursos materiales tales como bibliografía especializada en la enseñanza de la algoritmización mediante pseudocódigos o diagrama de flujo y fundamentos de lógica matemática.
- La **estructura** se conforma a partir de cuatro procedimientos didácticos que mantienen una sistemática interacción durante la dinámica de la algoritmización computacional, los que son denominados: construcción lógico – matemática, orientación matemático – algorítmica, estructuración algorítmico – generalizadora y validación algorítmico – computacional. Cada uno de estos procedimientos consta de dos tipos de acciones, uno dedicado a los profesores y el otro a los estudiantes. El sistema cuenta, además, con criterios evaluativos y patrones de logro para profesores y para estudiantes (ver anexo 9).

El procedimiento de la construcción lógico – matemática

Objetivo: Orientación a profesores y estudiantes sobre la forma de concretar, en el proceso de enseñanza – aprendizaje de la algoritmización para la resolución de problemas de programación computacional, las relaciones que se establecen entre la comprensión, interpretación y representación matemática de la situación problémica.

Acciones a realizar por el profesor

Durante su ejecución el docente debe propiciar la motivación de los estudiantes por la resolución de los problemas de programación, a partir de:

- Instruir al estudiante sobre algunos aspectos teóricos de suma importancia para abordar de forma consciente y preparada el proceso de algoritmización computacional.

  Mediante el método inductivo-deductivo el profesor puede llevar al grupo de estudiantes a "descubrir" qué es una situación problémica, para llegar a que reconozcan que la misma está formada por objetos (reales, matemáticos o computacionales), características de esos objetos y relaciones entre ellos, así como a visualizar las condiciones y exigencias que los agrupan. También debe llevarlos a comprender en qué consiste el proceso de resolución de una situación problémica mediante la algoritmización computacional, entre otros aspectos teóricos importantes.
- Considerar los conocimientos matemáticos previos para seleccionar las situaciones problémicas a utilizar.

  Se hace con el propósito de que los estudiantes aborden situaciones acordes a sus conocimientos previos, ganando en seguridad, al poder auto – valorar sus capacidades y posibilidades de resolverlas.
- Analizar situaciones problémicas que requieran de una representación matemática (por estar expresadas en términos de objetos y relaciones de la realidad), para motivar y transmitir patrones de actuación, a la vez que demostrar la utilidad de la programación, la que permite automatizar importantes procesos productivos, de servicio, etc.

Se puede lograr seleccionando situaciones problémicas provenientes de la esfera social, productiva y técnica del territorio y del país, a partir de las cuales el docente, mediante el método de elaboración conjunta, deberá mostrar la necesidad de leer varias veces la situación problémica, identificar los objetos reales contenidos en esta, sus características y relaciones, así como las condiciones y exigencias. Todo ello permitirá obtener una visión de sistema de la situación analizada, como condición previa a su interpretación matemática.

- Discutir numerosas situaciones problémicas sencillas, que vayan aumentando en su complejidad intrínseca y algorítmica.

  Crear las condiciones para que, mediante el trabajo en grupos, se realice el análisis minucioso de cada situación y se discuta colectivamente, con el objetivo de que se adquieran patrones para el desarrollo de dichos análisis, en aras de comprender las mismas al máximo antes de pasar a su resolución.

- Ejercitar la recuperación mental de conocimientos matemáticos básicos relacionados con la situación problémica bajo análisis.

  Con el propósito de promover la interpretación matemática de la situación problémica, deberá desarrollar actividades de elaboración conjunta en las que los estudiantes necesiten crear objetos matemáticos, asociados a los objetos reales que contiene la situación bajo análisis, así como asignarles características y datos guardados en su memoria.

- Demostrar cómo representar situaciones problémicas que hayan sido discutidas en el aula, a partir del uso de objetos y relaciones matemáticas conocidas.

  La intención de esta acción es lograr que el estudiante comprenda como integrar todos los elementos previamente analizados e interpretados en una primera representación matemática de la situación a resolver, la cual podrá exteriorizar mediante algún tipo de representación matemática como la conjuntista, numérica, algebraica, geométrica, tabular o analítica.

- Propiciar la confección de diversas representaciones matemáticas para un mismo problema.

  Esta acción se relaciona con la anterior, pero su objetivo es que una vez obtenidas esas diversas representaciones matemáticas, el estudiante pueda analizar cada una de ellas y compararlas para seleccionar la que le resulte más fácil de resolver, de acuerdo a las condiciones de la situación problémica y a sus conocimientos matemáticos. Este análisis permitirá inducir la necesidad de tener en cuenta que mientras más objetos y relaciones matemáticas contenga la representación seleccionada, mayor cantidad de estructuras lógico – computacionales serán necesarias para crear el pseudocódigo.

- Representar situaciones problémicas integradoras, que permitan la sistematización de la comprensión y la interpretación, desde los conocimientos y habilidades matemáticas.

  Puede hacerse empleando algún método problémico, integrando las acciones anteriores.

Acciones a realizar por el estudiante

Los estudiantes deben avanzar en la realización de acciones dinamizadoras del razonamiento lógico – matemático propuesto, es decir, dirigir sus acciones hacia:

- Crear el hábito de leer varias veces la situación problémica antes de comenzar a resolverla.

  Esto es para encaminar el análisis previo de la situación y evitar comenzar a trabajar con la primera idea que les viene a la mente, sin valorar las potencialidades de esta para el logro de una representación acertada.

- Interactuar con la situación problémica para obtener una primera comprensión del significado de la misma, para lo cual será necesario partir de:

  a) Realizar un análisis detallado de la situación problémica, mediante el cual se fragmente la misma en sus principales partes, para luego examinar minuciosamente cada una de ellas con el propósito de identificar los objetos (reales, matemáticos y/o computacionales) que las conforman, sus características, funciones y las relaciones que existen entre ellos.

b) Conectar los resultados del análisis y vincular las diversas partes de la información obtenida para establecer relaciones. Es decir; relacionar los datos similares y localizar las diferencias y semejanzas, para que, una vez hecha la comparación, se puedan establecer los vínculos entre los componentes de la información.

c) Determinar las partes más importantes o esenciales del objeto bajo estudio, a partir del análisis hecho, despreciando los rasgos y nexos secundarios, no determinantes del objeto.

- Activar sus conocimientos matemáticos y del contexto en que se desarrolla la situación problémica.

  Les permitirá ir imaginando objetos y relaciones, en correspondencia con los objetos y relaciones externas que presenta la situación que se está analizando.

- Integrar las partes seleccionadas, mediante el análisis, en un objeto más simplificado y esencial que el original, en cuanto a estructura y funciones, con lo que se obtendrían abstracciones de los objetos y relaciones matemáticas para obtener una interpretación de la situación problémica.

  Podrá lograrlo a partir de asociar a cada parte del objeto simplificado, objetos y relaciones matemáticas que lo identifiquen, de manera que obtenga una primera representación matemática de la situación problémica.

- Perfeccionar la representación matemática inicial.

  Este perfeccionamiento será posible volviendo sobre el análisis de la situación problémica, lo que le permitirá profundizar en esta y enriquecer la interpretación que de ella se había hecho. Esto podrá hacerse colectivamente, para que se transmitan patrones estrategias y métodos.

- Realizar varias representaciones matemáticas de un mismo problema.

  Esta forma de actuación le permitirá establecer comparaciones entre: el número de objetos utilizados, las relaciones establecidas y la intencionalidad deseada, lo que favorecerá la sistematización de los contenidos estudiados.

El procedimiento de la orientación matemático – algorítmica

Objetivo: Orientación a profesores y estudiantes sobre cómo llevar a la práctica, en el proceso de enseñanza – aprendizaje de la algoritmización para la resolución de problemas de programación computacional, las relaciones que se establecen entre la representación matemática de la situación problémica y la identificación e integración jerárquica de estructuras lógico – computacionales.

Acciones a realizar por el profesor

Para tener éxito en la ejecución de este procedimiento deberá favorecerse:

- Enseñar el pseudocódigo como herramienta para la definición de las estructuras lógico – computacionales.

  Hasta este momento el estudiante debe haber sistematizado las diferentes estructuras computacionales, sobre la base del lenguaje natural. Por lo que se hace necesario que el profesor seleccione o diseñe el pseudocódigo a emplear, explicándolo minuciosamente y basándose en numerosas ejemplificaciones.
- Trabajar en aras de la apropiación de conocimientos acerca de las propiedades y funciones de las estructuras lógico – computacionales.

  Puede lograrlo explicándoles dichas estructuras, sus funciones, relaciones, ventajas y limitaciones. Además de realizar una amplia ejemplificación de estos aspectos para que se fijen en la mente del estudiante y facilite así su posterior identificación. Se debe utilizar fundamentalmente el lenguaje natural de manera que le permita al estudiante captar la esencia de las estructuras sin desviar su atención en los aspectos técnicos de la sintaxis.
- Inducir a la utilización de diversas estructuras lógico – computacionales para un mismo problema.

  Se hace con el objetivo de analizar cada una de ellas y realizar comparaciones, teniendo en cuenta los objetos utilizados, la intencionalidad del uso de cada una, sus ventajas y limitaciones.
- Emplear métodos de enseñanza participativos para propiciar la reflexión y discusión de las representaciones de las situaciones problémicas.

Con esta acción se promueve la recuperación y reconstrucción de los conocimientos matemáticos y computacionales de los estudiantes.

- Incidir en que los estudiantes resuman los aspectos relevantes de la representación matemática bajo estudio.

  Esto tiene el propósito de suscitar la profundización en los objetos y relaciones que conforman dicha representación y facilitar que puedan recuperar de su mente estructuras lógico – computacionales adecuadas para obtener representaciones más esenciales de la misma.

- Facilitar el análisis de diferentes formas de ordenar y concatenar las estructuras computacionales, de acuerdo con las representaciones matemáticas obtenidas.

  Se hace con la intención de contribuir a formar en el estudiante patrones a seguir en la resolución de las situaciones problémicas de manera que el estudiante pueda valorar la pertinencia de cada concatenación e integración teniendo en cuenta las ventajas, desventajas, relaciones y características de cada estructura, explotando el uso de casos particulares para reforzar el aprendizaje de la representación matemática – computacional mediante estructuras algorítmicas, transitando de lo simple a lo complejo.

Acciones a realizar por el estudiante

Deberán dirigir su trabajo en la resolución de problemas de programación computacional hacia:

- Analizar la representación matemática de la situación problémica.

  Este análisis deberá irlo complementando con la interpretación de los objetos y relaciones que conforman dicha situación a partir de sus conocimientos computacionales.

- Identificar las estructuras algorítmicas necesarias para transformar los objetos y relaciones que aparecen en la representación matemática de la situación problémica.

  Por ejemplo, si está ante una representación matemática en términos de una sumatoria o productoria debe identificar como estructuras computacionales posibles a emplear, las iterativas (for, while, etc).

- Comparar las estructuras lógico – computacionales identificadas.

  Aquí, conocidas ya las principales estructuras computacionales básicas, deberá ser capaz de seleccionarlas, analizarlas y concatenarlas adecuadamente, en aras de formar una estructura funcional del proceso de programación y obtener niveles cada vez más esenciales y precisos de la representación matemática inicial, a partir de una perspectiva algorítmico – computacional.

- Sistematizar la representación de las estructuras lógico – computacionales identificadas e integradas en el pseudocódigo propuesto, a partir de representaciones matemáticas de situaciones problémicas más complejas.

  Esto puede hacerlo mediante el empleo de numerosas estructuras lógico – computacionales para remodelar diversas representaciones matemáticas, lo que les permitirá apropiarse de las principales vías de integración de dichas estructuras y alcanzar una adecuada familiarización con el pseudocódigo, antes de pasar al diseño del algoritmo. Cabe señalar que en algunas situaciones la representación matemática obtenida conduce a una modelación computacional muy compleja, lo que demanda una revisión y perfeccionamiento de la citada representación.

El procedimiento de la estructuración algorítmico – generalizadora

Objetivo: Orientación a profesores y estudiantes sobre la forma de concretar, en el proceso de enseñanza – aprendizaje de la algoritmización para la resolución de problemas de programación computacional, las relaciones que se establecen entre la generalización pseudocodificada de la representación matemática y la identificación e integración jerárquica de estructuras lógico – computacionales.

Acciones a realizar por el profesor

Para conseguir que dicha generalización sea eficaz deberá trabajar en el aula por:

- Lograr un compromiso, por parte del estudiante, hacia la construcción y sistematización de un amplio conocimiento sobre las formas de concatenar estructuras lógico – computacionales que se concretan en un algoritmo.

  Esto se puede lograr con la ejercitación de numerosas situaciones problémicas donde el estudiante deba identificar y concatenar adecuadamente estructuras lógico – computacionales. Aquí se le debe exigir que establezca comparaciones para justificar el uso de determinadas estructuras, así como su integración mediante un pseudocódigo en un algoritmo.

- Elaborar diferentes formas de estructurar una misma representación matemática de una situación problémica, usando pseudocódigos.

  Aquí se debe sistematizar el uso del pseudocódigo mediante la realización de numerosos ejemplos en los que se enfaticen las ventajas y desventajas de cada estructuración realizada.

- Desarrollar un convencimiento sobre lo imprescindible de aprender a construir algoritmos antes de introducirse en el conocimiento de un determinado lenguaje de programación.

  Se pueden mostrar ejemplos de las limitaciones e inconvenientes de hacer una estructuración computacional en un lenguaje de alto nivel (como C, C++, Java, C#, etc.) sin concebir un algoritmo previo. Luego comparar con la estructuración que se puede lograr usando un algoritmo en pseudocódigo, enfatizando en las ventajas de este proceder. Todo lo anterior permitirá explicitar los errores (estructurales y/o semánticos) que se pueden cometer cuando se pasa a resolver una situación problémica empleando directamente un lenguaje de programación.

- Reconocer y aplicar el pseudocódigo como herramienta base para la escritura del algoritmo y como vía de alcanzar la generalización en la solución de problemas de programación computacional.

  Con independencia de lo estrechamente relacionada que está esta acción con la anterior, es importante precisarla para puntualizar la necesidad de mostrar ejemplos de estructuración de algoritmos usando

pseudocódigo. En este punto deberán enfatizarse las ventajas del mismo, como herramienta que permite escribir algoritmos usando un lenguaje muy similar al lenguaje natural, lo que facilitará que el estudiante se identifique con el pseudocódigo de una forma más amena.

- Confrontar puntos de vista contrarios para contribuir a la elaboración de nuevas propuestas de estructuración que conduzcan a algoritmos más eficientes y generales.

  Esto conllevará a la modificación positiva de algunos patrones preestablecidos, además de fomentar en el estudiante hábitos de socialización de sus ideas y de aceptación del trabajo colectivo para perfeccionar sus propias ideas.

- Efectuar corridas del algoritmo que ha sido diseñado usando pseudocódigos.

  Las citadas corridas pueden llevarse a cabo de manera manual o mediante un software de apoyo como el PSeInt de O. Novara (2012), el RAPTOR de T. A. Wilson, M. C. Carlisle y J. W. Humphries (2012), el Software para la enseñanza – aprendizaje de algoritmos estructurados de J. J. Arellano y otros (2012), el Free DFP de F. Cárdenas, N. Castillo y E. Daza (2008), todos estos de acceso libre y multiplataforma. También puede utilizarse cualquier otro software que permita cumplir con el citado propósito. Esto favorecerá la creación y el análisis de nuevas generalizaciones algorítmicas, además de motivar al estudiante y desarrollar en él habilidades computacionales.

Acciones a realizar por el estudiante

Será preciso que encamine el proceso de aprendizaje de la resolución de problemas de programación computacional hacia:

- Realizar una resignificación y reestructuración lógico – computacional de la representación matemática de la situación problémica bajo estudio.

  Debe sistematizar la identificación, selección y concatenación de las estructuras lógico – computacionales, con el objetivo de afianzar los conocimientos adquiridos sobre las mismas.

- Obtener una generalización de la representación matemática, expresándola en pseudocódigos.

  Debe sistematizar la transformación de la representación matemática obtenida, en una estructuración algorítmica usando pseudocódigos. Esto con vista a obtener una primera aproximación de la solución algorítmica deseada.

- Ejercitar, manual o computacionalmente, la estructuración algorítmica de la representación matemática realizada usando pseudocódigos.

  Puede realizarla empleando el software indicado por el profesor, con el objetivo de desarrollar habilidades computacionales, a la vez que elevar la motivación por el estudio de la estructuración algorítmica mediante pseudocódigos.

- Aprovechar puntos de vista divergentes, emitidos por otros estudiantes, para perfeccionar sus propuestas de estructuración, de manera que sean más generales y eficientes.

  Esto conllevará a la modificación positiva de algunos de sus patrones de abordaje de la estructuración algorítmica, además de que potenciará sus hábitos de socialización de ideas y aceptación del trabajo en grupos. Esto además favorecerá la formación de habilidades para optimizar los algoritmos durante el proceso resolutor.

El procedimiento de la validación algorítmico – computacional

Objetivo: Orientación a profesores y estudiantes sobre la forma de concretar, en el proceso de enseñanza – aprendizaje de la algoritmización para la resolución de problemas de programación computacional, las relaciones que se establecen entre la generalización pseudocodificada de la representación matemática, la valoración sintáctica y semántica de la representación computacional y la valoración semántica de la representación computacional.

Acciones a realizar por el profesor

Para llevar a cabo este procedimiento será preciso:

- Considerar la importancia de que el estudiante domine las características del pseudocódigo utilizado, en relación con la definición del funcionamiento básico de las diferentes estructuras lógico – computacionales identificadas e integradas previamente.

  Este conocimiento básico se debe formar en el estudiante a partir de la sistematización de las características del pseudocódigo y su aplicación a la solución de numerosos ejemplos.

- Convencer al estudiante de que debe considerar como premisa esencial que el empleo de pseudocódigos no significa la ausencia de errores sintácticos.

  Mostrar al estudiante ejemplos concretos donde se evidencien errores sintácticos comunes al emplear pseudocódigos. Esto permitirá ir creando la necesidad de revisar minuciosamente el pseudocódigo del algoritmo una vez escrito, lo que contribuirá al desarrollo de habilidades para implementar algoritmos sintácticamente correctos en un lenguaje de programación.

- Trabajar en grupos para establecer discusiones que den la oportunidad de exponer y defender ideas sobre la intencionalidad de la estructuración realizada.

  Así se potenciará que los estudiantes se apropien de patrones de análisis y perfeccionamiento constante de los procesos de control sintáctico y de evaluación semántica.

- Evidenciar la importancia de optimizar el pseudocódigo para elevar la eficiencia y eficacia de los resultados.

  Esto puede lograrse a través de ejemplos concretos en los que se creen conflictos cognitivos que estimulen cambios en las estructuraciones algorítmicas realizadas, lo que favorecerá la formación de habilidades para diseñar algoritmos que permitan optimizar la memoria y el tiempo de ejecución del computador.

- Promover la ejercitación de la estructuración algorítmica realizada, tanto manual como empleando algún software que puede ser PSeInt, Free DFP, RAPTOR, el Software para la enseñanza – aprendizaje de

algoritmos estructurados u otro afín, con el objetivo de corroborar si el algoritmo cumple con la intencionalidad deseada.

La ejercitación debe permitir corroborar si los datos de salida del algoritmo presentan las características previamente estimadas para la solución de la situación problémica.

- Resolver ejemplos integradores que permitan sistematizar las acciones anteriores, potenciando la emergencia de un pensamiento algorítmico computacional.

  Esto facilitará que el estudiante observe el proceso resolutor completo y se apropie del modo de abordar una situación problémica, para lo cual debe de alcanzar niveles más esenciales y profundos de síntesis reflexivas, imprescindibles para conformar algorítmos que brinden soluciones eficientes y eficaces.

Acciones a realizar por el estudiante

En la dinámica del aula deberá encaminar su proceso de aprendizaje de la resolución de problemas de programación computacional hacia:

- Refinar el algoritmo, a partir de tomar en cuenta la escritura del pseudocódigo.

  Esto permitirá regular y evaluar sistemáticamente el proceso de algoritmización computacional con el objetivo de que el estudiante adquiera hábitos de revisión sintáctica de los algoritmos.

- Desarrollar habilidades de exactitud y precisión para el logro y control eficiente de los algoritmos.

  Será importante llevar al estudiante al convencimiento de que la exactitud y precisión de un algoritmo determina la calidad de su ejecución una vez implementado en un lenguaje de programación, dando respuesta a la solución de la situación problémica bajo estudio.

- Comprobar que el conjunto de sentencias sea completo.

  Debe verificar que el algoritmo contenga los pasos a realizar, suficientemente detallados, además de un conjunto de palabras reservadas del pseudocódigo concebido.

- Confirmar el flujo de control del algoritmo.

Se debe confirmar que el orden temporal en el cual se ejecutan los pasos individuales del algoritmo es correcto, es decir, que no se viola la disposición lógica de la secuencia de operaciones establecida.

- Rediseñar determinadas partes del algoritmo, en caso de que no funcionen bien.

  Es importante repetir el proceso de localización de errores, definiendo nuevos casos de prueba y recorriendo de nuevo el algoritmo con dichos datos.

- Valorar la validez de la representación computacional.

  La realización de una ejecución manual del algoritmo con datos significativos que abarquen todo el posible rango de valores para comprobar que la salida coincide con lo esperado en cada caso. Lo que facilitará ratificar si la estructuración del pseudocódigo es correcta en términos del contenido.

- Resolver problemas integradores, que estimulen la formación de un pensamiento algorítmico computacional.

  El estudiante transitará por el proceso resolutor completo y se apropiará de los patrones más relevantes que permiten abordar una situación problémica, conformando algorítmos que brinden soluciones eficientes y eficaces. Es importante estos problemas integradores contengan situaciones reales que requieran de un profundo análisis e interpretación por parte de los estudiantes, permitiendolo una sistematización amplia a través de los cuatro procedimientos del sistema.

Para medir la efectividad del sistema de procedimientos se proponen los criterios evaluativos y los patrones de logro para profesores y estudiantes involucrados en el proceso como se muestran a continuación.

Criterio evaluativo para los profesores: Pertinencia didáctica del proceso de enseñanza de la algoritmización para la resolución de problemas de programación computacional.

Patrones de logro para los profesores: Evidenciar que se ha logrado dar un tratamiento didáctico adecuado a la algoritmización en la resolución de problemas de programación computacional, desde el trabajo realizado para enseñar cómo:

No.	PATRONES DE LOGRO	CALIFICACIÓN			
		2	3	4	5
1	Analizar una situación problémica, de manera que se potencie la comprensión de la misma y su interpretación matemática.				
2	Realizar varias representaciones matemáticas de una situación problémica siempre que sea posible.				
3	Identificar las estructuras lógico – computacionales que sirven para transformar las representaciones matemáticas en pseudocódigos				
4	Integrar jerárquicamente las estructuras lógico – computacionales que sean seleccionadas para conformar el algoritmo.				
5	Realizar la valoración sintáctica de la estructuración algoritmica conformada mediante pseudocodigos				
6	Validar semánticamente el algoritmo realizando corridas manuales y/o empleando software especializados.				

Criterio evaluativo para los estudiantes: Desarrollo algorítmico alcanzado para la resolución eficiente y eficaz de problemas de programación como evidencia de un pensamiento algoritmico computacional.

Patrones de logro para los estudiantes: Mostrar la apropiación de habilidades para:

No.	PATRONES DE LOGRO	CALIFICACIÓN			
		2	3	4	5
1	La concepción de representaciones matemáticas de las situaciones problémicas que se les presenten.				
2	El reconocimiento de las estructuras algorítmicas que sirven para transformar las representaciones matemáticas en pseudocódigos, así como la determinación de las más apropiadas.				
3	La integración de las estructuras computacionales, previamente seleccionadas, en una generalización pseudocodificada de la representación matemática.				
4	El refinamiento de los algoritmos, a partir de la valoración de su sintaxis y su semántica.				
5	La validación y valoración de los algoritmos realizando corridas manuales y/o empleando software especializados.				

Estos criterios evaluativos y patrones de logro están concebidos para ser aplicados en cualquier momento del proceso de enseñanza – aprendizaje de la algoritmizacion para la resolucion de problema de programacion

computacional. En el caso de los profesores puede servir para la autoevaluación y la heteroevalución. Los resultados que se vayan obteniendo permitirán el perfeccionamiento sistemático del sistema de procedimientos didacticos.

Conclusiones

1. El modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional, que se propone, permitió revelar las configuraciones y relaciones esenciales entre los procesos que lo integran, dando lugar a que emergieran las dimensiones: construcción lógico – matemática, orientación matemático – algorítmica, estructuración algorítmico – generalizadora y validación algorítmico – computacional, que en su relación dialéctica potencian la formación de un pensamiento algorítmico computacional, imprescindible para la resolución eficaz y eficiente de los problemas de programación computacional.
2. La regularidad que se revela en el modelo propuesto está dada por la lógica integradora entre la representación matemática de la situación problémica y la generalización pseudocodificada de la representación matemática, la que desde el punto de vista didáctico rige el proceso formativo a desarrollar, encaminado al logro de un pensamiento algorítmico computacional.
3. El sistema de procedimientos didácticos está conformado por cuatro procedimientos, que se corresponden con las cuatro dimensiones de la dinámica modelada y permiten instrumentar el proceder lógico – algorítmico para la resolución de problemas de programación computacional, contando además con criterios evaluativos y patrones de logro para profesores y estudiantes.
4. La elaboración del modelo de la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional, que se concreta en el sistema de procedimientos didácticos propuesto, resuelve la contradicción dialéctica dada entre la representación matemática problematizada y su sistematización algorítmica.

CAPÍTULO III: CORROBORACIÓN Y VALORACIÓN CIENTÍFICA DE LOS PRINCIPALES RESULTADOS INVESTIGATIVOS

Introducción

En este capítulo se explican los resultados de la corroboración y valoración de la pertinencia científico – metodológica de los aportes teórico y práctico de la investigación. Se comienza presentando los resultados de cuatro talleres de socialización realizados, uno de ellos en la Universidad de las Ciencias Informáticas en La Habana, dos en carreras de ciencias computacionales de la Universidad de Oriente en Santiago de Cuba y uno en la carrera de Licenciatura en Ciencia de la computación de la Universidad Central "Marta Abreu" de las Villas, todos los cuales se acompañan de sus respectivos avales (ver anexo 10). Luego se muestra una ejemplificación del sistema de procedimientos didácticos y por último se expone un experimento pedagógico desarrollado en la carrera de Ingeniería en Telecomunicaciones y Electrónica de la Universidad de Oriente

3.1 Valoración de la factibilidad y pertinencia científico – metodológica de los principales resultados de la investigación a partir de la realización de talleres de socialización con especialistas

Para corroborar la factibilidad y pertinencia científico **–** metodológica del modelo y el sistema de procedimientos propuestos, se realizaron cuatro talleres de socialización con especialistas de tres universidades del país. Los criterios establecidos para la selección de los especialistas fueron:

- Categoría docente.
- Años de experiencia docente en la impartición de la Programación u otras asignaturas computacionales afines a la misma, tales como las pertenecientes a la Disciplina Principal Integradora.
- Experiencia en la aplicación de los contenidos de Programación a la solución de situaciones problémicas.

El primero de los citados talleres se realizó en el mes de marzo del año 2013, en la Universidad de las Ciencias Informáticas (UCI) de La Habana. Los participantes fueron 14 miembros del colectivo de la asignatura Programación de dicha universidad, de los cuales 9 son Profesores Auxiliares y 5 Asistentes. El promedio de años de experiencia docente en la impartición de la Programación fue de 8 años y el 100% tiene experiencia en la aplicación de los contenidos de Programación a la solución de situaciones problémicas.

El segundo taller se realizó en el mes de mayo del mismo año con los 4 profesores que conforman el colectivo de Programación para la carrera de Ingeniería en Telecomunicaciones y Electrónica, en la Universidad de Oriente, Santiago de Cuba. La categoría docente de estos profesores se distribuyó de la siguiente forma: 1 es Profesora Auxiliar y 3 Asistentes. El promedio de años de experiencia docente en la impartición de la Programación fue de 9 años y el 100% tiene experiencia en la aplicación de los contenidos de Programación a la solución de situaciones problémicas.

El tercer Taller se llevó a cabo en junio del año 2013 con un colectivo de 13 profesores que forman a los estudiantes de la carrera de Licenciatura en Ciencia de la Computación de la Universidad de Oriente, entre los que se incluye 1 Profesor Titular, 6 Profesores Auxiliares y 6 Asistentes. El promedio de años de experiencia docente en la impartición de la Programación fue de 13 años y el 100% tiene experiencia en la aplicación de los contenidos de Programación a la solución de situaciones problémicas.

El cuarto taller se llevó a cabo en marzo del 2014, ante 14 profesores del Departamento de Ciencia de la Computacíon de la Universidad Central “Marta Abreu” de las Villas, en el que participaron 8 Profesores Titulares, 2 Auxiliares, 3 Asistentes y 1 instructor. El promedio de años de experiencia docente en la impartición de la Programación fue de 15 años y el 100% tiene experiencia en la aplicación de los contenidos de Programación.

El **objetivo** de todos estos talleres fue la obtención de valoraciones que permitieran perfeccionar y enriquecer los resultados fundamentales de la investigación, a la vez que aportar algunos elementos didácticos que

sirvieran a los profesores del colectivo para abordar el compartido problema que con la algoritmización computacional enfrentan en el proceso de enseñanza – aprendizaje de la asignatura de Programación.

Los especialistas convocados contaron con una adecuada experiencia docente en el área de interés de la investigación. Esta fue la razón por la cual el debate alcanzó un alto nivel crítico – valorativo, lo que contribuyó de manera decisiva al perfeccionamiento de la propuesta investigativa.

El proceder metodológico llevado a cabo para desarrollar cada taller fue la realización de una presentación en la que el investigador explicó la modelación de la dinámica bajo estudio y el sistema de procedimientos didácticos para la algoritmización computacional. Luego solicitó intervenciones de los participantes en aras de obtener:

- Criterios válidos para el reforzamiento o la refutación de las bases teórico – metodológicas del modelo.
- Valoraciones críticas que sirvieran para el enriquecimiento del sistema de procedimientos didácticos a partir de la interpretación, las sugerencias y recomendaciones.
- Elementos valorativos que dieran cuenta de la factibilidad y pertinencia del sistema de procedimientos didácticos para ser introducido en la asignatura de Programación.

Los profesores participantes, durante el análisis y valoración de toda la información proporcionada, a la vez que esclarecían sus interrogantes, realizaron sugerencias y recomendaciones valiosas para el perfeccionamiento de la investigación, tal es el caso de las siguientes:

- Precisar lo que se ha asumido por “programación computacional”, a los efectos de la investigación.
- Considerar, en la modelación que se aporta, la discretización de la situación problémica una vez representada matemáticamente.
- Precisar el alcance del sistema de procedimientos de acuerdo a los contenidos de programación que se imparten, no debiéndose abarcar la resolución de cualquier problema de programación computacional.

- Pensar en introducir el sistema de procedimientos que se propone como un curso independiente, previo a la impartición de la asignatura de Programación, para que sirva de base a esta.
- Desarrollar ejercicios donde se resuelvan diversas situaciones problémicas de manera que se evidencie el paso por los cuatro procedimientos que propone el sistema.
- Considerar la posibilidad de incluir en el sistema de procedimientos didácticos el empleo de software que permitan la sistematización de los contenidos recibidos en el aula.
- Proponer un programa que refleje los contenidos y objetivos a impartir, así como el posible número de horas de duración del curso en el que se aplicarían los cuatro procedimientos del sistema.
- Usar pseudocódigos como alternativa para la representación algorítmica en vez de diagramas de flujos.
- Considerar la posibilidad de ampliar el uso del sistema de procedimientos didácticos que se propone a la enseñanza de la resolución de problemas matemáticos, teniendo en cuenta que los dos procedimientos iniciales del mismo permiten una correcta representación matemática.
- Realizar una corroboración del sistema de procedimientos didácticos en alguna de las carreras de ciencias computacionales empleando algún experimento pedagógico.

Todas estas recomendaciones realizadas fueron debidamente atendidas por el investigador y permitieron enriquecer y regular el proceso investigativo.

3.2 Ejemplificación del sistema de procedimientos didácticos

Con el objetivo de lograr una mayor comprensión del sistema de procedimientos didácticos y evidenciar su factibilidad de aplicación, se presentan dos ejemplos basados en situaciones problémicas (Salgado, A., Alonso, I. y Gorina, A., 2014a). Cabe precisar que estos dos ejemplos son ilustrativos del tipo de situaciones problémicas que se han usado en la aplicación parcial del sistema de procedimientos didácticos en el primer año de la carrera de Ingeniería en Telecomunicaciones y Electrónica, aplicación que se desarrolló en forma de experimento pedagógico y se explica en el epígrafe 3.3.

Ejemplo 1: Con el objetivo de mejorar las telecomunicaciones en la Universidad de Oriente, la Dirección de Informatización se propuso incluir en el año 2015 un nuevo servicio de conexión a Internet e intranet, usando tecnología inalámbrica (WiFi). El objetivo primario fue conectar las tres sedes de la universidad, Antonio Maceo Grajales, Julio Antonio Mella y Frank País García, para lo cual se necesitó comprar AP (Access Point) (Gralla, P., 2007) para redes WiFi. Es así que los especialistas se vieron ante la tarea de determinar qué tipos de AP comprar (según su alcance) y para ello necesitaron saber qué distancia existe entre las tres sedes. Además, al tener en cuenta que los comercializadores realizan una rebaja del 5% a cada dispositivo extra del mismo tipo que se les compre, se preguntaron si sería posible adquirir los tres AP necesarios al menor precio posible. Se pide diseñar un algoritmo que dé respuesta a la problemática anterior, conociendo las coordenadas (x_1, y_1), (x_2, y_2) y (x_3, y_3) de las tres sedes.

Observación: Esta situación problémica muestra un escenario aparentemente sencillo y fácilmente extrapolable al rol que debe desempeñar un profesional de las ciencias computacionales en la práctica, su transposición didáctico profesional busca despertar el interés del estudiante. Además, la misma está acorde a los conocimientos que deben tener los estudiantes que reciben por primera vez la asignatura de Programación en las carreras de ciencias computacionales, con lo cual se garantiza que los mismos ganen en seguridad y puedan auto – valorar adecuadamente sus capacidades y posibilidades resolutoras.

Procedimiento de la construcción lógico – matemática

Actuar del profesor: Debe dar tiempo a que cada estudiante, de manera individual, trate de comprender la situación problémica y luego propiciar un primer intercambio grupal en el que discutan la misma, de modo que se favorezca la interacción de lo individual con lo colectivo en el proceso de aprendizaje.

Actuar del estudiante: Analizar detalladamente la situación problémica que se describe y tratar de comprender cada uno de los elementos relevantes de la misma a partir de sus conocimientos matemáticos y del contexto profesional descrito, con lo que deberá identificar los objetos (algoritmo, coordenadas, distancia),

así como sus características, llegando a concluir que las coordenadas dadas forman un triángulo y que en dependencia del tipo de triángulo será el ahorro que se obtenga por concepto de precio de los AP (equilátero rebaja de un 10%, isósceles rebaja de un 5% y escaleno sin rebaja).

Posteriormente intentará expresar en lenguaje matemático cada uno de los objetos, características y relaciones relevantes que identifique en dicha situación y determinar las condiciones y exigencias, con lo que podría obtener que las condiciones del problema están dadas por las coordenadas de los tres vértices de un triángulo: Vértice_1 (x_1, y_1), Vértice_2 (x_2, y_2) y Vértice_3 (x_3, y_3) y que la exigencia consiste en determinar si el mismo es escaleno, isósceles o equilátero.

De lo anterior se deriva que la interpretación de la situación problémica puede llevarlos a comprender que necesitan diseñar un algoritmo que permita, dados los vértices de un triángulo (x_1, y_1), (x_2, y_2) y (x_3, y_3), determinar si el mismo es isósceles, equilátero o escaleno.

Observación: El estudiante deberá recuperar de su base de conocimientos y experiencias la información concerniente a la definición de cada tipo de triángulo según la longitud de sus lados, es decir, deberá identificar que un triángulo es:

- Escaleno: Si las longitudes de los tres lados son diferentes.
- Isósceles: Si las longitudes de dos lados cualesquiera son iguales.
- Equilátero: Si las longitudes de los tres lados son iguales.

Actuar del profesor: Debe inducir a los estudiantes a que realicen una representación gráfica de la situación problémica, empleando para ello los elementos básicos que hayan podido interpretar de las condiciones y teniendo en cuenta la exigencia.

Actuar del estudiante: Debe representar un triángulo con sus lados debidamente identificados, análogo a lo que se muestra en la Figura 3.2.1.

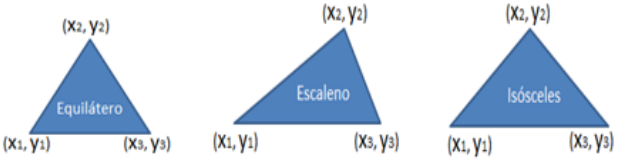


Figura 3.2.1: Representación geométrica del problema

Actuar del profesor: Debe orientar a los estudiantes hacia la búsqueda de relaciones entre los datos que propician las condiciones, para obtener una visión de sistema de la situación problémica; propiciando, por ejemplo, que los estudiantes recuperen de su mente información acerca de cómo se calcula la distancia entre dos puntos, enfatizando en la importancia de obtener las longitudes de los tres lados del triángulo, para facilitarles el establecimiento de comparaciones que les permitan determinar el tipo de triángulo.

Actuar del estudiante: Se espera que trate de representar las comparaciones de las tres longitudes o distancias de los lados del triángulo. Para esto deberá definir las variables que identificarán a esas tres distancias, lo que le permitirá el perfeccionamiento de la representación matemática inicial, que será posible volviendo sobre el análisis de la situación problémica para profundizar en ésta y enriquecer la interpretación que de ella se había hecho. La representación perfeccionada al considerar las tres distancias podría quedar como se muestra en la Figura 3.2.2.

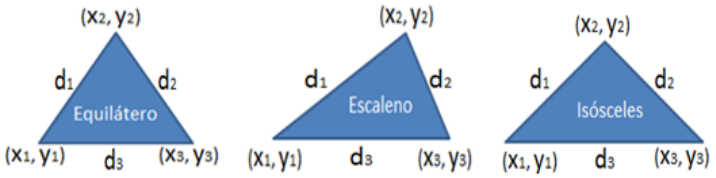


Figura 3.2.2: Representación geométrica transformada del problema

Al avanzar en la dinámica comparativa, animado por las preguntas e incentivos del profesor, podrá llegar a una representación más esencial, que es aquella que puede obtener a partir de objetos algebraicos, como se muestra en la Figura 3.2.3.

1. $d_1 = d_2 = d_3 \rightarrow$ **Equilátero**	2. $d_1 = d_2 \neq d_3 \rightarrow$ **Isósceles**
3. $d_1 \neq d_2 \neq d_3 \rightarrow$ **Escaleno**	4. $d_1 = d_3 \neq d_2 \rightarrow$ **Isósceles**
	5. $d_2 = d_3 \neq d_1 \rightarrow$ **Isósceles**

Figura 3.2.3: Representación algebraica del problema transformado

Actuar del profesor: Propiciar que el estudiante represente el problema de diversas formas, es decir, empleando objetos matemáticos de naturaleza geométrica, algebraica, entre otras. Para que pueda comparar dichas representaciones y seleccionar la más fácil, de acuerdo a las condiciones de la situación problémica y a sus conocimientos matemáticos. Así deberá aprovechar este análisis para inducir la necesidad de tener en

cuenta que mientras más objetos y relaciones matemáticas contenga la representación seleccionada, mayor cantidad de estructuras lógico – computacionales serán necesarias para crear el pseudocódigo.

Actuar del estudiante: Debe llegar a plantear la fórmula de la distancia entre dos puntos (distancia euclídea): $d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$; $i < j$, para posteriormente relacionar este conocimiento matemático con la representación anterior y llegar a una representación algebraica como la siguiente:

$$d_1 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2},\ d_2 = \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2},\ d_3 = \sqrt{(x_1 - x_3)^2 + (y_1 - y_{23})^2}$$

Observación: La relación entre este conocimiento matemático y la representación algebraica del problema, ilustrada en la Figura 3.2.3, puede ser considerada como la concreción del primer procedimiento y a la vez la base para iniciar el segundo.

El procedimiento de la orientación matemático – algorítmica

Actuar del profesor: Debe propiciar el debate sobre las estructuras computacionales que pueden ser empleadas para representar los objetos y relaciones matemáticas contenidas en la Figura 3.2.3, llevando a que comprendan que la representación lograda plantea tres situaciones a analizar: cuándo el triángulo es equilátero, cuándo es escaleno y en el caso de que éste sea isósceles, para lo cual será necesario el cálculo de las distancias d_1, d_2 y d_3. También será importante que precise el papel que juegan los vértices del triángulo para la introducción de las estructuras lógico – computacionales, cuyas coordenadas se constituyen en condiciones o datos de entrada. Deberá enfatizar en la necesidad del uso de pseudocódigos como herramientas para la definición de las citadas estructuras.

Actuar del estudiante: Tendrá que ir recuperando de su mente las estructuras algorítmicas necesarias para transformar los objetos y relaciones que aparecen en la representación matemática de la situación problémica (Figura 3.2.3). Se espera que identifique como una de las estructura el **"Si"** (if), teniendo en cuenta que tiene varias comparaciones que realizar entre las distancias de los lados del triángulo. No obstante, alguno podría identificar como estructuras a usar: **"**Mientras" (while) o "Desde" (for**)**. Sin

embargo, en el transcurso de la dinámica deberá llegar al convencimiento de que lo correcto es usar el **"Si"** pues, debido a las exigencias del problema y la representación realizada, no es necesario usar estructuras iterativas complejas como **"**Mientras**"** y **"**Desde**",** sino que solamente hay que realizar tres acciones fundamentales de comparación, las que por demás son distintas. A continuación se supone que analice y concatene las estructuras computacionales de manera adecuada, en aras de formar una estructura funcional esencial, a partir de lo cual podría obtener una estructura básica como se muestra en la Figura 3.2.4.

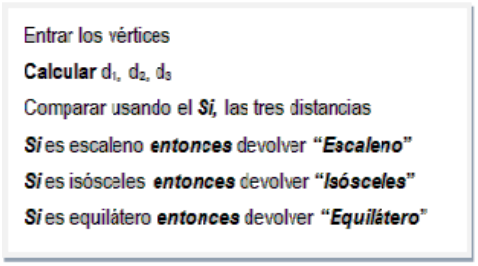


Figura 3.2.4. Integración básica de estructuras lógico–computacionales

Observación: La integración básica de estructuras lógico – computacionales, ilustrada en la Figura 3.2.4, se considera como la concreción del segundo procedimiento y a la vez la base para iniciar el tercero.

El procedimiento de la estructuración algorítmico – generalizadora

Actuar del profesor: Se asegurará de que los estudiantes realicen estructuraciones del algoritmo basadas en la integración obtenida, usando pseudocódigo y sistematizando las ventajas y desventajas de cada estructuración realizada. Para tales propósitos puede resultar provechoso el empleo de un software como el PSeInt, Free DFP, RAPTOR, el Software para la enseñanza – aprendizaje de algoritmos estructurados o cualquier otro afin que permita apoyar la dinámica resolutora.

Actuar del estudiante: Se espera que perfeccione la integración ilustrada en la Figura 3.2.4, al detallar cada paso descrito en la misma y analizar las posibles estructuraciones que conlleven a la solución.

Así, la estructuración algorítmica creada por el estudiante podría quedar como se ilustra en la Figura 3.2.5. Si bien la estructuración mostrada en la Figura 3.2.5 es válida, podrían proponer otra como en la Figura 3.2.6, utilizando los operadores lógicos (y/o). En ambas estructuraciones, por lo general, los estudiantes realizan la tercera comparación de las distancias, es decir escriben la sentencia que se muestra en la Figura 3.2.7.

Observación: Este paso es innecesario, puesto que si el triángulo no es equilátero ni escaleno, necesariamente tiene que ser isósceles.

Entrar los vértices (x_1, y_1) (x_2, y_2) (x_3, y_3)
Calcular d_1
$a_1 = (x_1 - x_2)\ (x_1 - x_2)$
$b_1 = (y_1 - y_2)\ (y_1 - y_2)$
$c_1 = a_1 + b_1$
$d_1 = \sqrt{c_1}$
Calcular d_2
$a_2 = (x_2 - x_3)\ (x_2 - x_3)$
$b_2 = (y_2 - y_3)\ (y_2 - y_3)$
$c_2 = a_2 + b_2$
$d_2 = \sqrt{c_2}$
Calcular d_3
$a_3 = (x_1 - x_3)\ (x_1 - x_3)$
$b_3 = (y_1 - y_3)\ (y_1 - y_3)$
$c_3 = a_3 + b_3$
$d_3 = \sqrt{c_3}$
Si $(d_1 \neq d_2)$ ***entonces***
{
Si $(d_1 \neq d_3)$ ***entonces***
{
Si $(d_2 \neq d_3)$ ***entonces***
{
"Escaleno"
}
}
Sino
{
"Isósceles"
}
Sino
{
Si $(d_1 == d_3)$ ***entonces***
"Equilátero"
Sino
"Isósceles"
}
}

Figura 3.2.5: Posible estructuración algorítmica.

Entrar los vértices (x_1, y_1) (x_2, y_2) (x_3, y_3)
Calcular d_1
$a_1 = (x_1 - x_2)\ (x_1 - x_2)$
$b_1 = (y_1 - y_2)\ (y_1 - y_2)$
$c_1 = a_1 + b_1$
$d_1 = \sqrt{c_1}$
Calcular d_2
$a_2 = (x_2 - x_3)\ (x_2 - x_3)$
$b_2 = (y_2 - y_3)\ (y_2 - y_3)$
$c_2 = a_2 + b_2$
$d_2 = \sqrt{c_2}$
Calcular d_3
$a_3 = (x_1 - x_3)\ (x_1 - x_3)$
$b_3 = (y_1 - y_3)\ (y_1 - y_3)$
$c_3 = a_3 + b_3$
$d_3 = \sqrt{c_3}$
Si $(d_1 \neq d_2)$ y $(d_1 \neq d_3)$ y $(d_2 \neq d_3)$ ***entonces***
{
"Escaleno"
}
Sino
{
Si $(d_1 == d_2)$ y $(d_1 == d_3)$ y $(d_2 == d_3)$ ***entonces***
{
"Equilátero"
}
Sino
{
"Isósceles"
}
}

Figura 3.2.6: Otra posible estructuración algorítmica.

Si $(d_1 == d_2)$ o $(d_1 == d_3)$ o $(d_2 == d_3)$ ***entonces***
{
"Isósceles"
}

Figura 3.2.7: Sentencia innecesaria.

Actuar del profesor: En este momento puede hacer preguntas sobre si es necesario o no hacer la última comparación y lo que implica computacionalmente el hecho de no hacerla, todo ello con el objetivo de ir formando en los estudiantes habilidades de optimización. Además pudiera introducir una modificación al problema, como por ejemplo la que consiste en suponer que además de determinar el tipo de triángulo se necesite saber cuál es el lado base (para el caso de ser isósceles). Aquí sí sería necesaria la tercera comparación, pues habría que saber cuáles son los lados iguales para concluir que el tercero es el lado base.

El procedimiento de la validación algorítmico – computacional

Actuar del profesor: Una vez estructurado el algoritmo que da solución a la situación problémica planteada, debe propiciar que se dominen las características del pseudocódigo, relacionadas con la definición del funcionamiento básico de las diferentes estructuras lógico – computacionales previamente identificadas e integradas. Todo ello con el propósito de que optimicen el pseudocódigo para elevar la eficiencia y eficacia de los resultados. Además debe provocar la discusión de las soluciones propuestas, para perfeccionarlas y eliminar errores de sintaxis, comparaciones innecesarias o errores que afecten la semántica de la solución.

Observación: En este momento del proceso resolutor se deben valorar las dos estructuraciones propuestas en las Figuras 3.2.5 y 3.2.6, llevando al estudiante a determinar cuál ofrece más ventajas computacionales en cuanto a optimizar la memoria y el tiempo de ejecución del computador.

Actuar del estudiante: Trabajará en el refinamiento sucesivo del algoritmo, a partir de tomar en cuenta la sintaxis o escritura del pseudocódigo. Esto le permitirá regular y evaluar sistemáticamente el proceso de algoritmización computacional. Deberá realizar una ejecución manual del algoritmo con datos completos, que abarquen todo el posible rango de valores, para comprobar que la salida coincide con lo esperado en cada caso. Se espera que verifique si el algoritmo contiene, con suficiente detalle, los pasos a realizar y el conjunto de palabras propias del pseudocódigo que ha sido previamente orientado. También podría rediseñar

determinadas partes del algoritmo sobre la base de valoraciones realizadas a otras estructuraciones obtenidas en la dinámica resolutora.

Entrar los vértices (x_1, y_1) (x_2, y_2) (x_3, y_3)
Calcular d_1
$a_1 = (x_1 - x_2)(x_1 - x_2)$
$b_1 = (y_1 - y_2)(y_1 - y_2)$
$c_1 = a_1 + b_1$
$d_1 = \sqrt{c_1}$
Calcular d_2
$a_2 = (x_2 - x_3)(x_2 - x_3)$
$b_2 = (y_2 - y_3)(y_2 - y_3)$
$c_2 = a_2 + b_2$
$d_2 = \sqrt{c_2}$
Calcular d_3
$a_3 = (x_1 - x_3)(x_1 - x_3)$
$b_3 = (y_1 - y_3)(y_1 - y_3)$
$c_3 = a_3 + b_3$
$d_3 = \sqrt{c_3}$
Si $(d_1 \neq d_2)$ ***entonces***
{
Si $(d_1 \neq d_3)$ ***entonces***
{
"Escaleno**"**
}
}
Sino
{
"Isósceles**"**
}
Sino
{
Si $(d_1 == d_3)$ ***entonces***
"Equilátero**"**
Sino
"Isósceles**"**
}
}

Figura 3.2.8: Estructuración más esencial respecto a la representada en la Figura 3.2.6

Entrar los vértices (x_1, y_1) (x_2, y_2) (x_3, y_3)
Calcular d_1
$a_1 = (x_1 - x_2)(x_1 - x_2)$
$b_1 = (y_1 - y_2)(y_1 - y_2)$
$c_1 = a_1 + b_1$
$d_1 = \sqrt{c_1}$
Calcular d_2
$a_2 = (x_2 - x_3)(x_2 - x_3)$
$b_2 = (y_2 - y_3)(y_2 - y_3)$
$c_2 = a_2 + b_2$
$d_2 = \sqrt{c_2}$
Calcular d_3
$a_3 = (x_1 - x_3)(x_1 - x_3)$
$b_3 = (y_1 - y_3)(y_1 - y_3)$
$c_3 = a_3 + b_3$
$d_3 = \sqrt{c_3}$
Si $(d_1 \neq d_2)$ y $(d_1 \neq d_3)$ ***entonces***
{
"Escaleno**"**
}
Sino
{
Si $(d_1 == d_2)$ y $(d_1 == d_3)$ y $(d_2 == d_3)$ ***entonces***
{
"Equilátero**"**
}
Sino
{
"Isósceles**"**
}
}

Figura 3.2.9: Estructuración más esencial respecto a la representada en la Figura 3.2.8

Actuar del profesor: Hacer notar que en la estructuración que ilustra la Figura 3.2.5 no es necesaria la comparación "***Si*** $(d_2 \neq d_3)$ ***entonces***" y en la Figura 3.2.6 cuando se emplea "***Si*** $(d_1 \neq d_2)$ y $(d_1 \neq d_3)$ y $(d_2 \neq d_3)$ ***entonces***", también sobra.

Actuar del estudiante: Debe llegar a una versión más refinada, como la que se ilustra en la Figura 3.2.9.

Actuar del profesor: Inducir un nuevo refinamiento de la solución pidiendo que profundicen en las sentencias dónde se determina que el triángulo es equilátero (Figura 3.2.9). Puede lograrlo a partir de preguntas como: ¿Creen ustedes que será necesario preguntar si **(d_2 == d_3)**? ¿Se obtiene el mismo resultado si se elimina esa comparación? ¿Cuál es la implicación en términos computacionales de eliminar la citada sentencia?

Ahora bien, para concluir la solución de la situación problémica planteada debe inducir el análisis de la rebaja del precio de los AP, a partir del tipo de triángulo obtenido con el algoritmo.

Observaciones finales: La situación problémica puede explotarse más introduciendo elementos de análisis que complejicen su algoritmización, tales como pedir a los estudiantes que incluyan en el algoritmo la verificación de que las coordenadas dadas forman un triángulo, agreguen en el algoritmo el cálculo del porciento de rebaja del precio de los AP, según el tipo de triángulo y se trabaje con los valores de los precios de los AP según su alcance para determinar el importe total.

Ejemplo 2: Considere una estación de servicios de correos electrónicos mediante acceso remoto con un único servidor. A la misma llegan peticiones provenientes de cuatro estaciones de trabajo. El servidor sólo atiende un cliente en cada momento y cuando termina se ocupa del que ha estado esperando más tiempo. Los clientes de cada estación pueden solicitar el acceso simultáneamente. Se requiere determinar todas las formas posibles de organizar la atención a los clientes cuando hay peticiones simultáneas, de manera tal que los mismos disfruten de iguales privilegios. Se pide crear un algoritmo para dar solución a esta situación.

Observación: Al igual que la primera situación problémica, la presente está relacionada con problemas inherentes al quehacer de los profesionales de las ciencias computacionales y su solución requiere de conocimientos que son asequibles a los estudiantes a los que va dirigida esta práctica docente.

El procedimiento de la construcción lógico – matemática

Actuar del profesor: Esperar que cada estudiante, de manera individual, se esfuerce por comprender la situación problémica y luego propiciar un primer intercambio grupal en el que discutan la misma.

A tales fines deberá hacer preguntas que lleven a que los estudiantes exploren la situación problémica, analizando cuidadosamente sus elementos y componentes, con la intención de crear patrones de búsqueda de una vía de solución. Debe conducir el proceso a que los estudiantes concluyan que en este caso son objetos dados: las estaciones de trabajo, el servidor y los clientes.

Actuar del estudiante: Analizar el texto de la situación problémica y tratar de comprender cada uno de los elementos relevantes de la misma, pues de esa comprensión dependerá que tenga éxito su resolución. Para la citada comprensión necesitará de conocimientos computacionales y del contexto, para interpretar la información contenida en el mismo. Se espera que los estudiantes imaginen las cuatro estaciones de trabajo como cuatro ordenadores conectados a un servidor principal. Además de que consideren que se hace una cola en espera de ser atendido, por lo que puede suceder que el mismo cliente siempre se atienda primero o al mismo cliente siempre le toque esperar. Llegando a la conclusión que se deben hallar todas las posibles formas de organizar o permutar los clientes para que todos puedan ser atendidos con igual prioridad.

Posteriormente, deberá tratar de expresar en lenguaje matemático cada uno de los objetos y relaciones que identifique en dicho texto, tratando de asociar una variable matemática a cada una de las actividades que conforman la situación, identificando las condiciones y las exigencias, con lo que podría obtener que las condiciones del problema están dadas por los cuatro objetos (en este caso clientes). Además de que la exigencia del problema consiste en calcular el factorial de 4.

Actuar del profesor: Conducir a los estudiantes hacia la búsqueda de las relaciones entre los datos, para obtener una visión de sistema de la situación problémica, que incida en el desarrollo del pensamiento algorítmico, permitiendo a los mismos encontrar mayor cantidad de implicaciones, nexos y relaciones dentro de la información, favoreciendo su visión sobre esta. Así mismo, debe propiciar que recuperen de su mente vías para el cálculo del factorial de un número. Esto permitirá al estudiante ir identificando las características de los objetos (matemáticos, reales, computacionales, entre otros) que están o podrían estar presentes en el

proceso de resolución de la situación problémica. Ello favorecerá el desarrollo de habilidades para seleccionar correctamente los objetos a utilizar, de acuerdo a las características identificadas.

Actuar del estudiante: Tratar de recuperar de su mente los conocimientos previos sobre el concepto de factorial de un número y concluir que es la productoria de todos los números menores que el número dado y por él mismo y que además el factorial de 0 es igual 1 por definición.

Actuar del profesor: Hacer énfasis en la importancia de representar u obtener los números menores que el número dado, puesto que harán falta para el cálculo del citado factorial.

Actuar del estudiante: Tratar de representar matemáticamente el cálculo del factorial de un número n cualquiera (en este caso 4). Para esto, en primer lugar deberá definir la variable que identificará al factorial y la definición del factorial de 0 como caso particular. También será necesario contemplar una variable para ir multiplicando en cada paso los números menores que n. Esto permitirá el perfeccionamiento de la representación matemática inicial.

Observación: El citado perfeccionamiento será posible volviendo sobre el análisis de la situación problémica, lo que permitirá profundizar en ésta y enriquecer la interpretación que de ella se había hecho. Esto podrá hacerse colectivamente, para que se transmitan patrones estrategias y métodos. Finalmente podría quedar así:

1. n: variable que representa el número al que se le calculará el factorial.
2. n! : Forma de representar matemáticamente el factorial de un número.
3. Por definición, 0! = 1.
4. Multiplicador: variable que tomará el valor de cada número menor que n, que se necesite multiplicar.

Actuar del profesor: Una vez obtenidas esas diversas representaciones matemáticas debe facilitar que el estudiante analice cada una de ellas y las compare, para seleccionar la que le resulte más fácil de resolver, de acuerdo a las condiciones de la situación problémica y a sus conocimientos matemáticos. Aquí se debe

aprovechar este análisis para insistir en que, mientras más objetos y relaciones matemáticas contenga la representación seleccionada, mayor cantidad de estructuras lógico – computacionales serán necesarias para crear el pseudocódigo.

Actuar del estudiante: Se espera en que plantee la fórmula para determinar el factorial de n. Posteriormente, deberá tratar de expresar en lenguaje matemático cada uno de los objetos y relaciones que identifique en dicho texto, tratando de asociar una variable matemática a cada una de las actividades que conforman la situación problémica, además de la concepción de abstracciones sustentadas en objetos y relaciones matemáticas para obtener una interpretación matemática de la situación problémica como se muestra a continuación. Esto podrá lograrlo a partir de asociar a cada parte del objeto simplificado, objetos y relaciones matemáticas que lo identifiquen, de manera que obtenga una primera representación matemática de la situación problémica como la que se muestra a continuación:

1. n! = n * (n – 1) * (n – 2) *…* 1 (Se calcula el factorial como el producto) (representación 1)
2. n! = 1 * 2 *…* (n – 1) * n (Se calcula el factorial como el producto) (representación 2)

Observación: Aquí el estudiante podría realizar varias representaciones matemáticas de la situación problémica, lo que permitirá establecer comparaciones entre el número de objetos utilizados, las relaciones establecidas y la intencionalidad deseada. Esto favorece la sistematización de los contenidos estudiados.

El procedimiento de la orientación matemático – algorítmica

Actuar del profesor: Propiciar el debate sobre las estructuras computacionales desde una perspectiva que permita reconocer las ventajas y desventajas de cada una, haciendo énfasis en que la representación lograda evidencia que el cálculo del producto requiere acumular las multiplicaciones de los valores n, n – 1, n – 2, etc. Debe hacer énfasis en la necesidad de leer primero el valor del número al que se le desea calcular el factorial, es decir, debe tener dicho número como entrada. Otro elemento importante es insistir en el uso del pseudocódigo como herramienta para la definición de las estructuras lógico – computacionales.

Actuar del estudiante: Ir reconociendo las estructuras algorítmicas necesarias para transformar los objetos y relaciones que aparecen en la representación matemática de la situación problémica. Convencerse de la necesidad de usar estructuras iterativas como el while, do while o el for, debido a que hay que realizar la misma acción de multiplicación varias veces de acuerdo a la representación matemática realizada y que por estas mismas razones no debe usar el "Si" (if).

Además deberá analizar y concatenar las estructuras adecuadamente para lograr niveles cada vez más esenciales y precisos de la representación matemática inicial desde de una perspectiva algorítmico – computacional. A partir de lo anterior se podría obtener una representación como la que se muestra en la Figura 3.2.10.

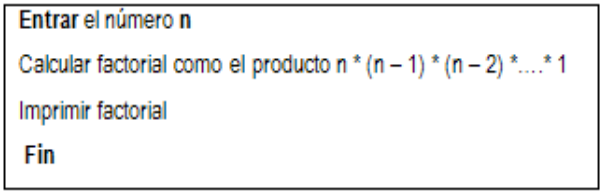


Figura 3.2.10: Integración básica de las estructuras lógico – computacionales.

Actuar del profesor: Hacer énfasis en las ventajas y desventajas de cada estructura lógico – computacional. Pudiendo formular preguntas tales como: ¿Cuáles son las características de las instrucciones while, do while, for? ¿Cuáles son sus ventajas? ¿Cuándo es mejor usarlas? ¿Por qué? Explicar que se podrá desarrollar la solución usando el while, do while o el for.

El procedimiento de la estructuración algorítmico – generalizadora

Actuar del profesor: Sugerir el empleo de un software como el PSeInt, Free DFP, RAPTOR o el Software para la enseñanza – aprendizaje de algoritmos estructurados. El uso de estas herramientas favorecerá la creación y análisis de nuevas generalizaciones algorítmicas, además de desarrollar habilidades computacionales. También deberá enfatizar en las ventajas del empleo de pseudocódigos.

Actuar del estudiante: Perfeccionar la representación anterior, detallando cada paso descrito en la Figura 3.2.9 y analizando las posibles estructuraciones que conlleven a la solución. Podría obtener una estructuración algorítmica usando el while como se muestra en la Figura 3.2.11.

Actuar del profesor: A partir de la Figura 3.2.11, podría realizar algunas preguntas al estudiante sobre la estructuración obtenida, como por ejemplo:

```
Estructuración usando while (Mientras Hacer)
Entrar el número n
factorial = 1 y multiplicador = n
mientras (multiplicador >1)
        {
           factorial = factorial * multiplicador
           multiplicador = multiplicador – 1
        }
Imprimir factorial
Fin
```

Figura 3.2.11: Posible estructuración algorítmica usando el while.

¿Qué pasaría si se decrementa el multiplicador antes de multiplicar por el factorial? ¿Se obtendría el mismo resultado? Es preciso explicar porqué no se debe decrementar el multiplicador antes de multiplicar por el factorial, ya que se estaría omitiendo la multiplicación por el número n dado. Esto suele ser un error común en este tipo de ejercicios. Si bien esta podría ser una estructuración válida, los alumnos podrían proponer otra utilizando el **do while** como se muestra en la Figura 3.2.12. Precisar que esta estructuración puede traer inconvenientes computacionales, pues primero se realiza la acción y luego se verifica la condición, que aunque en este caso no ocurre, como regla general sólo debe usarse cuando se pueda asegurar que las acciones a realizar se ejecutaran al menos una vez. Esto es una desventaja.

```
Estructuración usando do while (Hacer mientras)
Entrar el número n
factorial = 1
Si (n >1)
  {
     multiplicador = n
     Hacer
          {
             factorial = factorial * multiplicador
             multiplicador = multiplicador – 1
          }
     mientras (multiplicador >1)
  }
Imprimir factorial
Fin
```

Figura 3.2.12: Posible estructuración algorítmica usando el **do while**.

Actuar del estudiante: Puede obtener otra posible estructuración usando el **for**, como se muestra en la Figura 3.2.13. Por ejemplo, el cálculo del factorial de un número n es 1*2*…*n.

```
Estructuración usando for (Desde Hasta)
Entrar el número n
factorial = 1
      Desde multiplicador = 1 Hasta n multiplicador + 1
          {
             factorial = factorial * multiplicador
          }
Imprimir factorial
Fin
```

Figura 3.2.13: Posible estructuración algorítmica usando el for.

Observación: Precisar que hay que hacer n multiplicaciones para obtener el factorial (o ninguna, teniendo en cuenta que 0!=1).

Actuar del profesor: Realizar preguntas tales como: ¿Qué pasaría si se inicializara el multiplicador en 0? ¿Cambiaría el resultado? ¿Qué se podría hacer para que no cambie el resultado? ¿Cuántas iteraciones se van a realizar ahora? Precisar que la variable de control, que es el multiplicador, debe comenzar con el valor 1. Mientras ese controlador sea menor o igual a n (la cantidad de veces que el lazo debe iterar) se ejecutará el cuerpo del ciclo y se incrementará la variable de control. Hacer notar lo simple que es ahora la forma expresiva de la estructura para iterar. Precisar que como el multiplicador comienza con valor 1 y termina con valor n, entonces el lazo se ejecuta n veces (si el multiplicador comienza en 0, debe terminar en el valor n – 1, así se ejecutaría n veces).

Sistematizar cuándo es recomendable usar cada una de las estructuraciones obtenidas. Por ejemplo, la última estructuración con el for es viable siempre que se sepa la cantidad de veces que se va a repetir una acción, mientras que las dos primeras son más eficaces cuando no se tiene conocimiento del número de iteraciones a realizar y sólo se conoce una condición de parada o finalización del algoritmo.

El procedimiento de la validación algorítmico – computacional

Actuar del profesor: Observar que ya se tiene estructurado el algoritmo que da solución al problema, pero se debe hacer énfasis en que el estudiante domine las características del pseudocódigo utilizado, en relación con la definición del funcionamiento básico de las diferentes estructuras lógico – computacionales identificadas e integradas previamente. También debe retomar la importancia de optimizar el pseudocódigo para elevar la eficiencia y eficacia de los resultados. Discutir las soluciones propuestas para perfeccionarlas y eliminar errores de sintaxis, comparaciones innecesarias o errores que afecten la semántica de la solución.

Observación: En este momento se deben valorar las tres soluciones propuestas buscando cuál ofrece más ventajas computacionales en cuanto a optimizar la memoria y el tiempo de ejecución del computador. Es conveniente que la solución final se ejercite en un laboratorio con el software seleccionado.

Actuar del estudiante: Trabajar en el refinamiento del algoritmo, a partir de tomar en cuenta la sintaxis o escritura del pseudocódigo. Esto permitirá regular y evaluar sistemáticamente el proceso de algoritmización computacional. Comprender que la exactitud y precisión de un algoritmo determina la calidad de su ejecución, una vez implementado en un lenguaje de programación. Realizar una ejecución manual del algoritmo con datos significativos que abarquen todo el posible rango de valores para comprobar que la salida coincide con lo esperado en cada caso. Verificar que el algoritmo contenga los pasos a realizar, suficientemente detallados. Rediseñar determinadas partes del algoritmo.

Observación: En la estructuración usando el do while, por lo general los estudiantes no consideran dentro de la estructura del "**Si**" los elementos del do while y lo ponen al mismo nivel de identación o sangría.

Actuar del profesor: Explicar la importancia de respetar la sintaxis (el lenguaje natural que se esté usando según lo definido previamente) aunque se esté trabajado en pseudocódigo y escribir las estructuras que estén contenidas dentro de otras con mayor nivel de identación. Para concluir la solución de la situación problémica planteada debe inducir el análisis del resultado obtenido con el algoritmo que es 24 = 4 * 3 * 2 * 1, es decir, debe pedir a los estudiantes que una vez determinado el número de permutaciones o maneras distintas de organizar la atención de los cuatro clientes, propongan cómo sería esta disposición.

Actuar del estudiante: Se espera que en primera instancia se dé cuenta que al ser 24 permutaciones posibles, a cada cliente le corresponde ser el primero en atenderse 6 veces. Por lo que debe ir combinando todos los clientes hasta lograr el objetivo. En este momento se debe alcanzar una respuesta como la siguiente:

Cliente 1 con prioridad: C1 C2 C3 C4; C1 C2 C4 C3; C1 C3 C2 C4; C1 C3 C4 C2; C1 C4 C2 C3; C1 C4 C3 C2

Cliente 2 con prioridad: C2 C1 C3 C4; C2 C1 C4 C3; C2 C3 C1 C4; C2 C3 C4 C1; C2 C4 C1 C3; C2 C4 C3 C1

Cliente 3 con prioridad: C3 C2 C1 C4; C3 C2 C4 C1; C3 C1 C2 C4; C3 C1 C4 C2; C3 C4 C2 C1; C3 C4 C3 C1

Cliente 4 con prioridad: C4 C2 C3 C1; C4 C2 C1 C3; C4 C3 C2 C1; C4 C3 C1 C2; C4 C1 C2 C3; C4 C1 C3 C2

3.3 Aplicación parcial del sistema de procedimientos didácticos en el primer año de la carrera de Ingeniería en Telecomunicaciones y Electrónica de la Universidad de Oriente

Para llevar a cabo la corroboración se seleccionó la carrera de Ingeniería en Telecomunicaciones y Electrónica (ITE) de la UO, por ser esta una de las carreras de ciencias computacionales del centro y cumplir condiciones que hicieron propicio el desarrollo de un experimento pedagógico en el período en que la presente investigación transitaba por dicha etapa investigativa. Dentro de las condiciones se consideraron:

- Coincidencia, en un 80%, de los contenidos de la asignatura de Programación en los planes de estudio vigentes de las cuatro carreras de ciencias computacionales.
- El 66,53 % de los estudiantes de la carrera de ITE promovió con calificaciones de 3 puntos en dicha asignatura, en la etapa analizada, lo que da cuenta de la baja calidad de dicha promoción.
- A diferencias del resto de las carreras de ciencias computacionales de la UO, en el primer año de ITE existió una adecuada flexibilidad en la planificación docente, que facilitó que los estudiantes pudieran recibir el curso contemplado en el experimento en sesión contraria y con las frecuencias previstas.
- El primer año de ITE recibe la asignatura de Programación en el segundo semestre, coincidiendo en tiempo con la etapa de corroboración de la investigación.
- Existió una buena disposición del claustro de la carrera de ITE, así como de su dirección metodológica, para cooperar con el desarrollo del experimento.

El **objetivo** del experimento pedagógico fue la comprobación de la efectividad del sistema de procedimientos didácticos para potenciar el aprendizaje de la algoritmización en la resolución de problemas de programación.

Una vez seleccionada la carrera y el año, se diseñó y ejecutó un experimento pedagógico clasificado como **experimento con preprueba – postprueba y grupo control** (ver Hernández, R., Fernández, C. y Baptista, P., 1998, pp. 124 – 185). Este diseño incorporó la aplicación de prepruebas a los dos grupos que formaron parte el experimento (el grupo control y el experimental), ambos pertenecientes al primer año de la carrera de

ITE. Cabe precisar que para conformar los citados grupos, en lugar de que los estudiantes fueran asignados al azar a estos, se usó el emparejamiento como técnica que permite lograr una equivalencia entre los mismos, pues los dos grupos ya estaban conformados a la hora de llevar a cabo el experimento.

Inicialmente a los estudiantes de ambos grupos se les aplicó simultáneamente la preprueba. Luego un grupo recibió el tratamiento experimental (grupo experimental) y otro no lo recibió (grupo control). Finalmente se les aplicó a ambos grupos una postprueba, también simultáneamente. El diseño empleado puede diagramarse como muestra la Tabla 3.3.1.

Tabla 3.3.1: Diagrama del diseño del experimento pedagógico con preprueba – postprueba y grupo control.

EG_1	**O_1**	**X** Sistema de procedimientos didácticos	**O_2**
EG_2	**O_3**	**---** Método Tradicional	**O_4**

E: Emparejamiento o técnica de apareo (en inglés matching).
G_1, **G_2**: Grupo experimental y grupo control respectivamente.
O_i (i=1, 2, 3, 4): Una medición a los estudiantes de un grupo (prueba pedagógica). Si aparece antes del estímulo se trata de una preprueba, si es después una postprueba.
X: Condición experimental (aplicación del sistema de procedimientos didácticos). Presencia de algún nivel de la variable independiente.
--- Ausencia de estímulo (nivel cero en la variable independiente, lo que equivale a desarrollar la dinámica del proceso de enseñanza – aprendizaje de la programación por la vía tradicional). Indica que se trata de un grupo control.

El empleo de la preprueba ofreció dos ventajas:

- Las puntuaciones de las prepruebas se usaron para fines de control en el experimento. Al compararse las prepruebas de los dos grupos se pudo evaluar cuán correcto fue el emparejamiento.
- Se pudo analizar el puntaje o ganancia de cada grupo (la diferencia entre las puntuaciones de la preprueba y la postprueba).

Este diseño tuvo en cuenta las principales fuentes de invalidación interna, por lo que la administración de la prueba quedó controlada. Si la preprueba hubiese afectado las puntuaciones de la postprueba, los resultados de dicha afectación hubiesen sido similares en ambos grupos, de aquí que se siguiera cumpliendo con la esencia del control experimental. En general, durante el experimento las variables que influyeron en un grupo

también lo hicieron de la misma manera en el otro, salvo para el caso de la variable experimental, esto permitió mantener la equivalencia de los grupos. Así, durante la realización del experimento se controló: a) la "historia" como fuente de invalidación interna, pues no ocurrió ningún acontecimiento significativo que afectara a los grupos (control y experimental), b) la "maduración" porque al existir un emparejamiento de los estudiantes de ambos grupos, la maduración esperada fue similar en los mismos, c) la "inestabilidad" en la aplicación de los instrumentos y en su medición, pues se aplicaron en iguales condiciones y tiempo y fueron calificados por un único profesor y d) la "administración" de pruebas, pues la preprueba no afectó las puntuaciones de la postprueba, ya que hubo suficiente tiempo entre la aplicación de las mismas. A continuación se muestran los resultados obtenidos en cada una de las fases del experimento pedagógico.

Emparejamiento y aplicación de la preprueba

Se aplicó la preprueba que se muestra en el anexo 11.1 y se calificó la prueba en base a la escala ordinal (2, 3, 4, 5). Luego, para cada uno de los 23 estudiantes de cada grupo (control y experimental) se ordenaron las calificaciones de menor a mayor y se procedió a realizar el emparejamiento (ver Tabla 3.3.2). Se pasó a analizar si existían diferencias significativas entre el grupo experimental y el grupo control respecto a su tendencia central, tomando como base las calificaciones obtenidas. El caso de existencia de diferencias significativas es equivalente a que los estudiantes de cada grupo hayan sido extraídos de poblaciones diferentes (independencia entre grupos), de lo contrario, se concluye que los estudiantes de ambos grupos provienen de una misma población.

Tabla 3.3.2: Resultados de la aplicación de la preprueba y emparejamiento en los dos grupos (experimental y control). Fuente, elaboración propia.

Est.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	PT
GExp.	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	4	5	**59**
GCon.	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3	4	4	**59**

A tales efectos se utilizó la Prueba no paramétrica U de Mann – Whitney (ver Siegel, S., 1972, pp. 143 – 155), que puede ser utilizada para al menos una escala de tipo ordinal y permite docimar que las muestras provienen de igual población o poblaciones diferentes. Las hipótesis formuladas fueron:

- **H_o**: Las calificaciones de la preprueba del grupo control y experimental no difieren significativamente respecto a su tendencia central (mediana).
- **H_A**: Las calificaciones de la preprueba del grupo control y experimental difieren significativamente respecto a su tendencia central (mediana).

El nivel de significación que se utilizó fue $\alpha = 0,05$ y se docimó mediante la aproximación a la normal (prueba de dos colas) para valores mayores que 20, obteniéndose que $Z = – 0,21$ por lo que no hay evidencia para rechazar **Ho**, pudiéndose concluir que ambas muestran pertenecen a una misma población, o bien, que el proceso de emparejamiento llevado a cabo fue correcto.

Aplicación del tratamiento, variable independiente X (Sistema de procedimientos didácticos)

El sistema de procedimientos didácticos (variable independiente X) se aplicó en un subgrupo de 23 alumnos del primer año de la carrera de ITE (considerado como grupo experimental), durante un periodo de tiempo de 18 semanas del año 2013. La frecuencia semanal fue de dos encuentros de dos horas cada uno, es decir, de 4 horas semanales, para un total de 72 horas presenciales, a las que se adicionaron otras 72 horas no presenciales. Cabe precisar que tanto el grupo control como el experimental recibieron los mismos contenidos de la asignatura de Programación y que el segundo (experimental) recibió, además, clases de algoritmización computacional, basadas en las acciones del sistema de procedimientos didácticos que se propone, y empleando problemas análogos a los explicados en el epígrafe anterior. El otro grupo no recibió la citada docencia de algoritmización computacional.

Para el experimento se tomó como variable dependiente: **Y** $\rightarrow$ aprendizaje de la algoritmización para la resolución de problemas de programación computacional. Considerando que dicho aprendizaje está en

correspondencia con los cuatro procedimientos del sistema propuesto, así como con los criterios evaluativos y patrones de logro que aporta el mismo.

Aplicación de la postprueba

Se aplicó la postprueba pedagógica para evaluar la efectividad del sistema de procedimientos (ver anexo 11.2), obteniéndose las calificaciones que se presentan en la Tabla 3.3.3. Con el propósito de conocer si había diferencias significativas entre las calificaciones obtenidas por el grupo experimental y el grupo control se utilizó la Prueba U de Mann – Whitney**,** para muestras no relacionadas, ideal para el caso de una escala de medición de tipo ordinal. A tales efectos se plantearon las siguientes hipótesis:

- **H_o**: Las calificaciones de los estudiantes del grupo experimental son menores o iguales que las del grupo control en la postprueba.
- **H_A**: Las calificaciones de los estudiantes del grupo experimental son mayores que las del grupo control en la postprueba.

Tabla 3.3.3. Resultados de la postprueba. Fuente elaboración propia.

Est.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	PT
GExp.	4	5	4	2	4	4	5	4	5	2	3	2	3	4	5	4	3	4	3	3	5	3	5	86
GCon.	5	4	3	2	4	2	2	3	3	2	4	2	3	4	3	3	3	4	2	3	2	3	4	73

El nivel de significación que se utilizó fue α = 0,05 y se docimó (para una prueba de una cola) mediante la aproximación a la normal, obteniéndose que Z = – 2,25, por lo que se rechaza **Ho.** De lo anterior se pudo concluir que hay suficientes evidencias en los datos obtenidos para plantear que existen diferencias significativas entre las calificaciones de los estudiantes del grupo control y experimental en la postprueba, siendo estas últimas significativamente mayores, lo que implica un mejor aprovechamiento académico. Se concluye así, desde el experimento pedagógico realizado (centrado en el perfeccionamiento de la dinámica del proceso de algoritmización), que el sistema de procedimientos didácticos que se aporta brinda suficientes evidencias sobre su influencia positiva en el perfeccionamiento de la variable dependiente: **Y** → aprendizaje de la algoritmización para la resolución de problemas de programación computacional.

La concreción de ese aprendizaje se evidenció en un incremento de habilidades para concebir representaciones matemáticas de las situaciones problémicas que el profesor propuso, así como para identificar, seleccionar e integrar las estructuras algorítmicas necesarias para transformar estas representaciones, obteniendo generalizaciones basadas en pseudocódigos. También se observó un notable avance en el refinamiento de los algoritmos, a partir de la valoración de su sintaxis y su semántica y en el desarrollo de corridas de los mismos.

Conclusiones

1. Se valoró la pertinencia científica del modelo de la dinámica lógico – algorítmica de la resolución de problemas de programación computacional y la factibilidad de aplicación del sistema de procedimientos didácticos que se proponen, a través del desarrollo de cuatro talleres de socialización con especialistas, los que arrojaron un resultado favorable y permitieron enriquecer y perfeccionar el citado sistema, reconociéndose el valor epistémico y praxiológico de estos aportes.
2. La ejemplificación del sistema de procedimientos didácticos permitió ilustrar la intencionalidad investigativa del autor, al concretar el instrumento práctico a casos particulares, resultando de gran utilidad para la implementación de la experimentación desarrollada con posterioridad, al mostrar una forma viable de emplearlo para perfeccionar la dinámica lógico – algorítmica del proceso de resolución de problemas de programación computacional.
3. El experimento pedagógico realizado en el primer año de la carrera de Ingeniería en Telecomunicaciones y Electrónica de la Universidad de Oriente brindó suficientes evidencias para concluir que el sistema de procedimientos didácticos que se propone está en condiciones de perfeccionar el aprendizaje de la algoritmización en la resolución de problemas de programación computacional, aportando elementos de su factibilidad de aplicación en carreras de ciencias computacionales.

CONCLUSIONES GENERALES

1. La fundamentación epistemológica y praxiológica del objeto y el campo de acción de esta investigación, así como el análisis de sus tendencias históricas, revelaron insuficientes referentes teóricos y metodológicos del proceso de enseñanza – aprendizaje de la resolución de problemas de programación computacional que permitieran profundizar en el proceso de algoritmización, en particular, desde la relación de integración que puede establecerse entre la representación matemática problematizada y su sistematización algorítmica, lo que se constituyó en el camino hipotético que condujo la construcción del conocimiento científico que se propone.
2. El modelo revelado y fundamentado es expresión de la lógica integradora entre la representación matemática de la situación problémica y su generalización pseudocodificada, como condición imprescindible para el desarrollo de un pensamiento algorítmico computacional en los estudiantes, relación que a la vez da cuenta de la doble modelación (matemática y computacional) que debe experimentar una situación problémica en el camino algorítmico que conduce a su solución.
3. La lógica integradora entre las dimensiones del modelo se concreta en un sistema de procedimientos didácticos, el que favorece y orienta a profesores y estudiantes de Programación en la conducción de la dinámica del proceso de enseñanza – aprendizaje de la algoritmización computacional.
4. La reconstrucción epistemológica realizada sobre el objeto de estudio y el campo de acción de la investigación, se sintetizó en los aportes teórico y práctico de la tesis, los que se caracterizan por un adecuado nivel de abstracción del objeto de investigación transformado y por una profunda coherencia lógica. Ello sustenta la esencialidad de dichos aportes, que fueron valorados y corroborados a través de la realización de cuatro talleres de socialización con especialistas, la ejemplificación del sistema de procedimientos didácticos y el desarrollo de un experimento pedagógico, evidenciándose resultados satisfactorios.

RECOMENDACIONES

Para los investigadores en Didáctica de la Programación:

1. Continuar profundizando en el estudio del proceso de algoritmización computacional para develar y fundamentar un sistema de habilidades básicas que contribuya a la orientación formativa de la algoritmización computacional, explicitando cada una de estas habilidades y ejemplificando su empleo, para que sirva de guía a los docentes de Programación.
2. Trabajar en la concepción y elaboración de un método algorítmico computacional, que como constructo teórico – práctico, ilustre la correspondencia entre la modelación teórica y el sistema de procedimientos didácticos que se propone.
3. Diseñar e implementar sistemas computacionales que automaticen aspectos relevantes de la nueva dinámica propuesta.

Para las direcciones metodológicas y administrativas de las carreras de ciencias computacionales:

4. Viabilizar la introducción de los aportes de la presente investigación como contenido de cursos previos a la asignatura de Programación, en las carreras de ciencias computacionales y extenderlos a otras carreras que contemplen dicha asignatura, para que sirvan de sustento a los contenidos que deben impartirse en las mismas y contribuyan a elevar su asimilación y la calidad de los resultados docentes que se alcancen.
5. Garantizar la impartición de cursos de postgrado para profesores de Programación, con la intención de facilitar la apropiación de los elementos principales de la modelación que se propone y del sistema de procedimientos didácticos.
6. Garantizar la elaboración de materiales didácticos (programa analítico, guía metodológica, etc.) que posibiliten la orientación de los profesores de Programación de las carreras de las ciencias computacionales para la utilización de los aportes de la tesis.

REFERENCIAS BIBLIOGRÁFICAS

1. ACM´13. (2013). ACM / IEEE – CS Computer Science Curricula 2013. Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. Disponible en: http: //www.computer.org/education/cc2005/ironman/cc2005/index.html. [Consultado el 12 de Enero de 2014].
2. ACM´08. (2008). ACM / IEEE – CS Computer Science Curricula 2008. An Interim Revision of CS 2001. Report from the Interim Review Task Force. Disponible en: http://www.computer.org/education/cc2008/ironman/ cc2008/index.html. [Consultado 1 de Junio, de 2013].
3. ACM´05. (2005). ACM / IEEE – CS Computing Curricula 2005. The Overview Report. Disponible en: http://www.computer.org/education/cc2005/ironman/cc2005/index.html. [Consultado el 8 de Marzo, de 2013].
4. ACM´01. (2001). ACM / IEEE Computing Curricula 2001 – DRAFT. Disponible en: http://www.computer.org/education/cc2001/ironman/cc2001/index.html. [Consultado el 5 de Julio, de 2011].
5. ACM´91. (1991) ACM / IEEE – CS Computing Curricula 1991. Report of the ACM/IEEE – CS Joint Curriculum Task Force ACM Press and IEEE Computer Society Press, 1991. Disponible en: http://www.acm.org/education/curr91/ homepage.html. [Consultado el 2 de Febrero, de 2011].
6. ACM´78. (1978). Curriculum Committee on Computer Science. Curriculum 78: Recommendations for the undergraduate program in computer science. Communications of the ACM, 22(3):147 – 166, 1979.
7. ACM´68. (1968). Association for Computing Machinery, Curriculum '68: Recommendations for academic programs in computer science, In ACM Curricula recommendations for computer science, Association for Computing Machinery, 1968.
8. Aho, A. V., Hopcroft, J. E. y Ullman, J. D. (1998). Estructuras de datos y algoritmos. Con la colaboración de Guillermo Levine Gutiérrez. Versión en español de Américo Vargas y Jorge Lozano. Publicación México, DF: Addison – Wesley.

9. Alberto, M., Rogiano, C., Roldán, G. y Banchik, M. (2009). Fortaleciendo las habilidades matemáticas de los alumnos ingresantes desde los entornos virtuales. Disponible en: http://www.soarem.org.ar/Documentos/39%20Alberto.pdf [Consultado el 19 de Enero de 2011].
10. Alexandrescu, A. (2010). The D Programming Language. Addison – Wesley. ISBN 978-0-321-63536-5.
11. Alfonseca, M. (1974). SIAL / 71, a Continuous Simulation Compiler, in Advances in Cybernetics and Systems, Ed. J. Rose, Gordon and Breach, London, Vol. 3, 1974, 1319 – 1340.
12. Al – Imamy, S., Alizadeh, J. y Nour, M. A. (2006). On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process. In Journal of Information Technology Education. Vol. 5, Year 2006. pp. 271 – 283.
13. Alonso, I. (2001). La resolución de problemas matemáticos. Una alternativa didáctica centrada en la representación. Tesis en Opción al Grado Científico de Doctor en Ciencias Pedagógicas, Universidad de Oriente, Santiago de Cuba, Cuba.
14. Arellano, J. J. y Nieva, O. S. (2009). Método de enseñanza de algoritmos centrado en 2 dimensiones. 4to Simposio Internacional en Sistemas Telemáticos y Organizaciones Inteligentes. SITOI 2009. pp. 881 – 897. ISBN: 978–607–95043–2–8. Xalapa Veracruz, México. Disponible en: http://www.unistmo.edu.mx/~jjap/id26_SITOI.pdf [Consultado el 5 de Diciembre de 2012].
15. Arellano, J. J., Nieva, O. S., Solar, R. y 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, Diciembre 2012, pp. 23 – 33.
16. Arellano, N., Fernández, J., Rosas, M. V. y Zúñiga, M. E. (2014). Estrategia metodológica de la enseñanza de la programación para la permanencia de los alumnos de primer año de Ingeniería Electrónica. En Revista Iberoamericana de Educación en Tecnología y Tecnología en Educación (TE & ET). No. 13, Junio 2014, pp. 55 – 60.

17. Aspray, W. (1990). John von Neumann and the Origins of Modern Computing. (MIT Press, 1990, ISBN: 0262011212, 9780262011211) Disponible en: http://books.google.com/books/about/JohnVonNeumannandtheOriginsofModer.html?id=c5uDQgAACAAJ. [Consultado el 10 de Diciembre de 2012].
18. Ausubel, D. P. (2002). Adquisición y retención del conocimiento. Una perspectiva cognitiva. Barcelona: Paidós.
19. _____ (1976). Psicología educativa. Un punto de vista cognoscitivo. México: Trillas.
20. _____ (1973). Algunos aspectos psicológicos de la estructura del conocimiento. En: Elam, S. (Comp.) La educación y la estructura del conocimiento. Investigaciones sobre el proceso de aprendizaje y la naturaleza de las disciplinas que integran el currículum, (pp. 211 – 239). Buenos Aires: El Ateneo.
21. Backus, J., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Naur, P., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., Wijngaarden A. V. y Woodger, M. (1963). Revised Report on the Algorithmic Language ALGOL 60. Commun. ACM, Vol. 6, No. 1, pp. 1–17.
22. Barchini, G., Sosa, M. y Herrera, S. (2004). La informática como disciplina científica. Ensayo de mapeo disciplinar. En Revista de Informática Educativa y Medios Audiovisuales. Año 1, Volumen 1, Número 2. Disponible en: http://laboratorios.fi.uba.ar/lie/Revista/articulos.htm. [Consultado el 5 de Junio de 2012].
23. Bayona, J. A. y otros. (2005). Actitudes frente a la venta y el consumo de sustancias psicoactivas al interior de la universidad nacional de Colombia. En Interamerican Journal of Psychology, Sociedad Interamericana de Psicología. vol. 39, número 001, pp. 159 – 168.
24. Blanco, L. (2003). Apuntes para una historia de la Informática en Cuba. Universidad de la Habana.
25. Brennan, K. y Resnick, M. (2013). Aprender a programar, programar para aprender. Disponible en: http://edtk.co/ly676 [Consultado el 3 de Enero de 2014].
26. Bueno, E. (1976). Lógica polivalente. Editorial. Ciencias Sociales. La Habana, 1976.
27. Caignaert, C. (1988). Étude de l'évolution des méthodes d'apprentissage et de programmation. Bulletin de L'EPI, 50, 52 – 60.

28. Castro, E. y Castro, E. (1997). Representaciones y modelización. Cap. IV del libro: La Educación Matemática en la Enseñanza Secundaria. Barcelona. Editorial Horsori. Luis Rico (Cor.). pp. 95 – 124.

29. Cetín, I. (2013). Visualization: a tool for enhancing students' concept images of basic object – oriented concepts. En Revista Computer Science Education, 23:1, pp. 1 – 23.

30. Chesñevar, C. I. (2001). Utilización de los mapas conceptuales en la enseñanza de la programación. Disponible en: http://cs.uns.edu.ar/~cic/2000/2000-jornadas-mapas/2000-jornadas-mapas.pdf [Consultado el 10 de diciembre de 2011].

31. Cochran, W. G. (1980). Muestreo. Trillas, 1980.

32. Cormen, T. H., Leiserson, C., Rivest, R. y Stein, C. (2009). Introduction to algorithms. Cambridge, Massachusetts: The MIT Press. ISBN 978 – 0 – 262 – 53305 – 8.

33. Cuevas, R. E., Miranda, A. y Martínez, J. M. (2011). Actualización del Plan de Estudios del Ingeniero en Computación usando el ME y A, bajo el enfoque constructivista de la UAGro. Disponible en: http://www.somece.org.mx/Simposio2011/index.php?option=com_content&view=article&id=10&Itemid=16. [Consultado el 10 de Enero de 2012].

34. Davis, H. C., Hugh, C. y White, S. (2011).The personalisation of a learning environment: student-led connections online and offline. At HEA Enhancement Academy Team Leaders Meeting in May 2011, University of Southampton, 25 – 26 May 2011. Disponible en: http://eprints.soton.ac.uk/272349/. [Consultado el 5 de diciembre de 2011].

35. Delgado, J. R. (2000). Las habilidades generales matemáticas. Disponible en: http://www.soarem.org.ar/Documentos/Actas%20de%20la%20VII%20Carem.pdf [Consultado el 25 de Junio de 2011].

36. De Lobos, M. E. (2010). Aprende a programar. Disponible en: http://www.mailxmail.com/curso-aprende-programar/tipos-estructuras-programacion-estructuras-basicas-secuencial. [Consultado el 24 de Febrero de 2011].

37. Denning, P. J. (2000). Computer Science: The Discipline (PDF). Encyclopedia of Computer Science. Disponible en: http://www.idi.ntnu.no/emner/dif8916/denning.pdf. [Consultado el 20 de Abril de 2012].

38. Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A. B., Turner, A. J. y Young, P. R. (1989). Computing as a discipline. Communications of the ACM, 32 (1):9 – 23, January 1989.

39. Dillon, E., Anderson, M. y Brown, M. (2014). Teaching Students to Program Using Visual Environments: Impetus for a Faulty Mental Model? En Journal of Computational Science Education. Volume 5, Issue 1, August 2014.

40. Duval, R. (1999). Semiosis y pensamiento humano. Registros semióticos de aprendizajes intelectuales. Universidad del Valle de México.

41. _____. (1993). Semiosis et Noesis. Lecturas en Didáctica de la Matemática: Escuela Francesa.

42. Expósito, C. (1995). Enfoques actuales en la enseñanza de la Informática. Ponencia Presentada en la Jornada Científica metodológica. ISPEJV. 1995.

43. Expósito, C. y otros. (2001). Algunos elementos de metodología de la enseñanza de la informática. Universidad de la Habana.

44. Faouzia, B. y Mostafa, H. (2007). Utilisation des NTICs pour l'apprentissage et l'autoévaluation de l'algorithmique. SETIT 2007, 4th International Conference: Sciences of Electronic, Technologies of Information and Telecommunications. TUNISIA, Marzo 25 – 29, 2007.

45. _____ (2006). Utilisation des exemples et de la démonstration dans l'Apprentissage de l'Algorithmique. Disponible en: http://hal.archives-ouvertes.fr/docs/00/11/28/21/PDF/BenHan_2006.pdf. [Consultado el 12 de Junio de 2010]

46. Fariñas, J. L. (2009). Modelo de la dinámica de formación del pensamiento algorítmico singularizado en las consultas SQL en alumnos de los Politécnicos de Informática. Tesis presentada en opción al grado científico de Doctor en Ciencias Pedagógicas. ISP, Enrique José Varona, La Habana, Cuba.

47. Fergusson, E. M., Salgado, A., Alonso, I. y Gorina, A. (2015). Consideraciones epistemológicas sobre la formación investigativa del licenciado en Ciencia de la Computación. En Revista Órbita Pedagógica, Año 2015, Vol. 2, No. 2 (Mayo – Agosto), pp.45 – 68.

48. Ferreira, A. y Rojo, G. (2005). Enseñanza de la programación. En Revista Iberoamericana de Tecnología en Educación y Educación en Tecnología. Año 2005, Volumen 1, Número 1 Disponible en: http://teyet-revista.info.unlp.edu.ar/numero-1.htm [Consultado el 25 de Junio de 2013].

49. _____. (2005). Cambios metodológicos – didácticos y evaluación del impacto de los mismos en un curso introductorio a los conceptos de algorítmica y programación. Primeras Jornadas de Educación en Informática y TICS en Argentina. JEITICS 2005, pp. 210 – 216.

50. Fuentes, H. C. (2009). Pedagogía y Didáctica de La Educación Superior. En CD ROM "Aportes del CeeS a la Pedagogía Moderna". Memoria Científica II [ISBN: 978 – 959 – 207 – 385 – 2].

51. _____. (2000). Didáctica de la Educación Superior. Santa Fe de Bogotá, Colombia: INPAHU.

52. Fuentes, H. C., Matos, E. C. y Cruz, S. S. (2004). El Proceso de Investigación Científica desde un Pensamiento Dialéctico Hermenéutico. Reto actual en la formación de doctores. Libro inédito CeeS "Manuel F. Gran". Universidad de Oriente, Santiago de Cuba, Cuba.

53. Fuentes, H. C., Montoya, J. y Fuentes, L. (2012). La formación en la Educación Superior: Desde lo holístico, complejo y dialéctico de la construcción del conocimiento científico. Universidad Técnica de Esmeraldas "Luis Vargas Torres" y Universidad de Oriente, Santiago de Cuba. Ediciones Mutile 2012.

54. Gagné, E. D. (1991). La psicología cognitiva del aprendizaje escolar. Madrid, España: Ed. Visor Distribuciones, S. A.

55. Goldstine, H. H. y Goldstine, A. (1982). The Electronic Numerical Integrator and Computer (ENIAC). The Origins of Digital Computers: Selected Papers. New York: Springer – Verlag, pp. 359 – 373. ISBN 3 – 540 – 11319 – 3.

56. González, M. C. y Paniagua, J. G. (2010). Interpretación de problemas matemáticos. Disponible en: http://davidbuiles.files.wordpress.com/2011/01/interpretacic3b3n-de-problemas-matemc3a1ticos.pdf [Consultado el 12 de Mayo de 2011].

57. González, W. (2001). Hacia un enfoque sistémico en la enseñanza de la Informática. Evento Internacional COMAT 2001. Universidad de Matanzas.

58. González, W., Estrada, V. y Martínez, M. (2006). Contribución al desarrollo de la creatividad a través de la enseñanza de la programación. Revista digital Pedagogía Universitaria. Vol. 9. No.3. Disponible en: http://169.158.24.166/texts/pd/1894/04/3/189404308.pdf [Consultado el 12 de Mayo de 2010].

59. _____. (2006). Metodología para contribuir al desarrollo de la creatividad en los estudiantes de la educación superior a través de la enseñanza de la programación. Memorias del Congreso Internacional. La ciencia y el humanismo en el siglo XXI: Perspectivas. Universidad Iberoamericana de la Ciudad de México, los días 31 de Marzo a 2 de abril de 2006.

60. Gorina, A. (2010). Dinámica del procesamiento de la información en las investigaciones sociales. Tesis presentada en opción al grado científico de Doctor en Ciencias Pedagógicas, Centro de Estudios de Educación Superior "Manuel F. Gran", Universidad de Oriente, Cuba.

61. Gorina, A., Alonso, I., Salgado, A. y Álvarez, J. A. (2014). La gestión de la información científica proporcionada por el criterio de expertos. En Revista Cubana de Ciencias de la Información Vol. 45, No. 2, Mayo – Agosto, pp. 39 – 47, 2014.

62. Gorina, A., Alonso, I. y Zamora, L. (2010). La indagación y la gestión estadística de datos en el proceso de investigación científica de las Ciencias Sociales. En CD ROM Aportes del CeeS "Manuel F. Gran" a la Pedagogía Moderna. Memoria Científica II [ISBN: 978 – 959 – 207 – 385 – 2].

63. Gorina, A. y Alonso, I. (2012). Un sistema de procedimientos metodológicos para perfeccionar el procesamiento de la información en las investigaciones sociales. En Revista Didasc@lia, Volumen 3, No. 6 (Diciembre) (Monográfico especial), pp. 91 – 108.

64. Guibert, N., Guittet, L. y Girard, P. (2005a). A study of the efficiency of an alternative programming paradigm to teach the basics of programming. Disponible en: http://www.lisi.ensma.fr/fr/equipes/idd/publications.html [Consultado el 10 de Enero de 2012].

65. _____. (2005b). Initiation à la Programmation par l'exemple: concepts, environnement, et étude d'utilité. Environnements Iinformatiques pour l'Apprentissage Humain (EIAH), Montpellier, France, 2005, pp. 461 – 466. Disponible en: http://telearn.archives-ouvertes.fr/hal-00005762 [Consultado el 21 de Abril de 2011].

66. _____. (2006). Performances et usages d'un environnement d'apprentisage de la programmation basé sur exemple. ERGO'IA, 2006, pp. 103 – 110.

67. Gralla, P. (2007). Cómo funcionan las redes inalámbricas. Anaya Multimedia. ISBN 978-84-415-2068-4.

68. Grier, D. A. (2005). When computers were human. Princeton University Press. ISBN 84 – 89660 – 00 – X. Disponible en: http://web.archive.org/web/20060525195404/ o http://www.idi.ntnu.no/emner/dif8916/grier.pdf. [Consultado el 20 de Abril de 2012].

69. Halvorson, M. (2008) Visual Basic 2008 (1a ed. edición). Anaya Multimedia. pp. 656 ISBN 978 – 84 – 415 – 2448 – 4. Disponible en: http://portal.acm.org/citation.cfm?id=20411. [Consultado el 4 de Abril de 2010].

70. Hernández, R., Fernández, C. y Baptista, P. (1998). Metodología de la investigación social. McGraw – HILL Interamericana Edit., S. A. de C. V. [ISBN: 970 – 10 – 1899 – 0].

71. Hodges, A. (1983). Alan Turing: The Enigma, Vintage edition 1992, first published by Burnett Books Ltd, 1983. ISBN 0 – 09 – 911641 – 3.

72. Johnson, S. C. y Kernighan, B. W. (1983). The C Language and Models for Systems Programming, BYTE, pp. 48 – 60, August 1983.

73. Joyanes, L. (2004). Fundamentos de programación, Estructuras de Datos y Objetos. McGraw Hill, España, 3ra edición, 2004. ISBN: 84 – 481 – 3664 – 0.

74. _____. (1998). Fundamentos de Programación. Algoritmos y Estructura de Datos. McGraw Hill, España, 1998. ISBN: 84 – 481 – 0603 – 2.

75. Käasboll, J. (2002). Learning Programming. University of Oslo.

76. _____. (1998). Exploring didactic models for programming: Norsk Informatikk – konferanse, Høgskolen i Agder. Disponible en: http://www.sciencedirect.com/science/article/pii/S187704281103014X. [Consultado el 7 de Octubre de 2010].

77. Kemeny, J. G. y Kurtz, T. E. (1986). Structured BASIC programming. New York, USA: John Wiley & Sons. ISBN 0 – 471 – 81087 – 8. Disponible en: http://portal.acm.org/citation.cfm?id=20411. [Consultado el 4 de Abril de 2010].

78. Kinnunen, P. y Simon, B. (2012). My program is ok – am I? Computing freshmen's experiences of doing programming assignments. En Revista Computer Science Education, Vol. 22, No. 1, pp. 1 – 28.

79. Knowlton, J. (2009). Python. Anaya Multimedia – Anaya Interactiva. ISBN 978 – 84 – 415 – 2513 – 9.

80. Knuth, D. E. y Pardo, L. T. (1980). The early development of programming languages. A history of computing in the twentieth century (a collection of essays). London: Academic Press, 1980.

81. Kordakia, M., Miatidisb, M. y Kapsampelisa, G. (2008). A computer environment for beginners' learning of sorting algorithms: Design and pilot evaluation. En Revista Computers & Education, Vol. 51, No. 2, Septiembre 2008, pp. 708 – 723.

82. Kurkovsky, S. (2013). Mobile game development: improving student engagement and motivation in introductory computing courses. En Revista Computer Science Education, Vol. 23, No. 2, pp.138 – 157.

83. Labarrere, A. (1994). Pensamiento, Análisis y autorregulación en la actividad cognoscitiva de los alumnos. Angeles Editores. México.

84. Lee, J. A. N. (1995). Computers Pioneers. IEEE Computer Society Press. Los Alamitos, California.

85. Levine, G. (2001). Computación y programación moderna. Perspectiva integral de la informática. Pearson Educación, México, 2001. ISBN: 968 – 444 – 485 – 0. p. 640.

86. Lissabet, A. y Cruz, M. A. (2011). Evolución histórica del proceso de formación de la cultura informática del profesor de computación en la educación cubana. En Revista Cuadernos de Educación y Desarrollo, Año 2011, Volumen 3, no.23 (Enero).

87. Ma, L., Ferguson, J., Roper, M. y Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. En Revista Computer Science Education. Volumen 21, Número 1, 2011, pp. 57 – 80.

88. Machado, M. (2000). La enseñanza – aprendizaje de los Procesadores de Textos en el Preuniversitario (una alternativa metodológica sobre la base del Sistema Integrado Works). Tesis en opción al título de Master en Informática Educativa. Mención Enseñanza de la Informática. Instituto Superior Pedagógico Enrique José Varona. Cuba.

89. Martinelli, O. (2006). Elementos básicos de programación en C. Ediciones AKAL. S.A., Madrid.

90. Martínez, J. A. y Martínez, L. (2008). Determinación de la varianza máxima para el cálculo del factor de imprecisión sobre la escala de medida, y extensión a diferentes tipos de muestreo. En Revista Psicothema, vol. 20, número 002, pp. 311 – 316.

91. Martínez, M. (1999). El desarrollo de la creatividad mediante la enseñanza problémica en la actualidad. Teoría y práctica. Pedagogía 99.

92. Martínez, S. y Fariñas, J. L. (2012). La competencia elaborar programas informáticos desde el proceso de enseñanza – aprendizaje de la disciplina lenguaje y técnicas de programación. En Revista Didasc@lia, Año 2012. Volumen. 3. Número 2, Abril – Junio, pp. 125 – 144.

93. Martínez, Y. (2005). En busca de una nueva forma de enseñar a programar. Investigación bibliográfica. Disponible en: http://www.mty.itesm.mx/rectoria/dda/riee/pdf05/27(DTIE).YolandaMtz.pdf [Consultado el 4 de Abril de 2012].

94. Mateu, M. M. (1998). Un sistema para evaluar el Programa de Informática Educativa. Tesis presentada en opción al título académico de máster en informática educativa. Instituto Superior Pedagógico Enrique José Varona. La Habana.

95. Matos, E. C. y Cruz, L. (2011). La práctica investigativa, una experiencia en la formación doctoral en Ciencias Pedagógicas. Libro CeeS “Manuel F. Gran”. Universidad de Oriente, Cuba.

96. Metcalf, M., Reid, J. y Cohen, M. (2004). Fortran 95 / 2003. Explained 3e. Oxford University Press, Oxford, England.

97. Miños, A. M. (2015). Didáctica de la informática y métodos formales: ¿Por qué son importantes los métodos formales para la didáctica de la informática? En Revista Electrónica Formación y Calidad Educativa (REFCalE). Vol. 3, Año 2015, No. 1 (Enero – Abril). pp. 105 – 116.

98. Moroni, N. y Señas, P. (2004). Aplicación de mapas conceptuales hipermediales en la visualización de programas. Disponible en: http://cmc.ihmc.us/papers/cmc2004-252.pdf [Consultado el 7 de Enero de 2011].

99. _____. (2005). Estrategia para la enseñanza de la programación. Disponible en: http://cs.uns.edu.ar/jeitics2005/Trabajos/pdf/52.pdf [Consultado el 4 de Abril de 2012].

100. Murillo, M. (2006). Explorando el proceso de enseñanza y de aprendizaje en el área de la programación de computadoras En Revista Electrónica Actualidades Investigativas en Educación Año 2006, Volumen 6, Número 1, ISSN 1409 – 4703, pp. 1 – 28.

101. Neyret, R. (2005). Situation problème et programmation, Revue Grand N n°37, CRDP Grenoble.

102. Norberg, A. L. (2005). Computers and Commerce: A Study of Technology and Management at Eckert–Mauchly. Computer Company, Engineering Research Associates and Remington Rand, 1946 – 1957. The MIT Press. ISBN 0 – 262 – 14090 – X.

103. Novara, P. (2012). PSeInt. Disponible en: http://pseint.sourceforge.net/ [Consultado el 10 de Septiembre de 2014].

104. Oviedo, M. y Ortiz, F. G. (2002). La enseñanza de la programación. Disponible en: http://bibliotecadigital.conevyt.org.mx/colecciones/documentos/somece2002/Grupo4/Oviedo.pdf [Consultado el 25 de Abril de 2012].

105. Pérez, R. (2009). Una herramienta y técnica para la enseñanza de la programación. Disponible en: http://campusv.uaem.mx/cicos/imagenes/memorias/6tocicos2008/Articulos/Cartel%206.pdf [Consultado el 13 de Septiembre de 2010].

106. Peyton, J. y Sheard, T. (2002). Template meta – programming for Haskell, Proceedings of the Haskell Workshop, Pittsburgh. Disponible en: http://research.microsoft.com/en-us/um/people/simonpj/Papers/papers.html. [Consultado el 13 de Septiembre de 2011].

107. Pinales, F. J. y Velázquez, C. E. (2014). Algoritmos resueltos con diagramas de flujo y pseudocódigo. ISBN: 978 – 607 – 8285 – 96 – 9. Universidad Autónoma de Aguascalientes, México. Disponible en: http://www.uaa.mx/direcciones/dgdv/editorial/docs/algoritmos.pdf [Consultado 1 de enero de 2015].

108. Planchart, O. (2005). La Modelación Matemática: alternativa didáctica en la enseñanza de precálculo. Revista de Investigación en Ciencias Matemáticas, Volumen 1, Junio de 2005.

109. Polya, G. (1965). Cómo plantear y resolver problemas, Trillas, 1965.

110. Pozo, J. I. (1994). La solución de Problemas. Santillana. Aula XXI, Madrid, 1994.

111. Ramírez, R. V. (1991). NEWT, una herramienta de programación gráfica para la enseñanza del pensamiento algorítmico. IX Reunión de Intercambio de Experiencias en Estudios sobre Educación. Monterrey, N.L., México, Agosto de 1991.

112. Ramos de Melo, F., Flôres, E. L., Diniz de Carvalho, S., Gonçalves de Teixeira, R. A., Batista, L. F. y Renato de Sousa, G. (2014). Computational organization of didactic contents for personalized virtual learning environments. En Revista Computers & Education. Volumen 79, Octubre 2014, pp. 126 – 137.

113. Randall, A. (2006). Una entrevista perdida con el co – inventor de ENIAC, J. Presper Eckert. Computer World. Disponible en: http://es.wikipedia.org/w/index.php?title=ENIAC&amp. [Consultado el 25 de Abril de 2011].

114. Remedios, M. A. (2006). La lógica de programación en los joven club de computación y electrónica. Disponible en: http://www.monografias.com/trabajos41/joven-club-computacion/joven-club-computacion2.shtml [Consultado el 20 de Mayo de 2012].

115. Reynolds, C. y Tymann, P. (2008). Schaum's Outline of Principles of Computer Science. McGraw – Hill. ISBN 978 – 0 – 07 – 146051 – 4.

116. Rico, L., Castro, E. y Romero, I. (1996). The Role of Representation Systems in the learning of Numerical Structures. En A. Gutiérrez y L. Puig (edts.). Proceedings of the Twentieth International Conference for the Psychology of Mathematics Education Vol.1. Valencia: Universidad de Valencia.

117. Ritchie, D. M. (1993). The Development of the C Language. Disponible en: http://cm.bell-labs.com/cm/cs/who/dmr/chist.html. [Consultado el 20 de Mayo de 2012].

118. Rodríguez, M. (2013). Dinámica del proceso de enseñanza – aprendizaje de la computación desde una lógica interdisciplinar. Tesis presentada en opción al grado académico de Máster en Ciencias de la Educación Superior. Universidad de Oriente, Santiago de Cuba, Cuba.

119. Salazar, C. y Delrieux, C. (2004). Asignaturas introductorias a la programación: una discusión acerca de sus objetivos y contenidos programáticos. XXIX Conferencia Latinoamericana de Informática, CLEI, 29 Septiembre al 2 de Octubre 2004, La Paz, Bolivia. Disponible en: http://lsm.dei.uc.pt/ribie/docfiles/txt20031212151824TCI12.pdf [Consultado el 20 de Febrero de 2011].

120. Salett, M. y Hein, N. (2004). Modelación matemática y los desafíos para enseñar matemática. Revista: Educación Matemática, 16(2). ISSN impreso: 1665 – 5826. Disponible en: http://www.redalyc.org/pdf/405/40516206.pdf [Consultado el 28 de Diciembre de 2013].

121. Salgado, A., Alonso, I. y Gorina, A. (2014a). Ejemplificación de la solución algorítmica de problemas de programación computacional. En Revista Didasc@lia: Didáctica y Educación. Vol. 5, No. 4. Año 2014, (Octubre – Diciembre), pp. 15 – 36.

122. _____. (2014b). Sistema de Procedimientos Didácticos para perfeccionar la Algoritmización Computacional. 1ra Conferencia Científica Internacional UCIENCIA 2014. Universidad de las Ciencias Informáticas. Cuba. Disponible en: https: //uciencia.uci.cu/sites/default/files/public/p2926-ponencia-529_0.pdf [Consultado el 20 de Mayo de 2014].

123. Salgado, A., Alonso, I. y Gorina, A. (2013). Aproximación a la resolución de problemas de programación computacional y su lógica de algoritmización. En Revista Santiago, Año 2013, No. 131 (Mayo – Agosto), pp. 519 – 531.

124. Salgado, A., Alonso, I., Gorina, A. y Tardo, Y. (2013a). Lógica algorítmica para la resolución de problemas de programación computacional: una propuesta didáctica. En Revista Didasc@lia, Año 2013, Volumen 4, No. 1 (Enero – Marzo), pp. 57 – 76.

125. _____. (2013b). Didáctica de la Resolución de Problemas de Programación Computacional. En Revista Pedagogía Universitaria. Vol. XVIII No. 4, pp. 62 – 74.

126. Salgado, A., Gorina, A. y Alonso, I., (2013). Modelo de la dinámica lógico – algorítmica para la resolución de problemas de programación computacional. En Revista Educare, Año 2013, Vol. 17, No. 1 (Enero – Abril), pp. 27 – 51.

127. Santiesteban, Y. (2013). Estrategia educativa para la formación del valor de la perseverancia en la resolución de problemas matemáticos. Tesis presentada en opción al grado académico de Máster en Ciencias de la Educación Superior. Universidad de Oriente, Santiago de Cuba, Cuba.

128. Santos, L. (1995). La transferencia en el uso del conocimiento: un paso necesario en el aprendizaje de las matemáticas. IX Reunión Centroamericana y del Caribe. Ciudad de la Habana. Cuba.

129. Sebesta, R. W. (2012). Concepts of Programming Languages. Tenth Edition. University of Colorado at Colorado Springs, 2012. Disponible en: www.PlentyofeBooks.net. [Consultado el 25 de Enero 2013].

130. Selby, D. (2002). Jottings from the business intelligence jungle. In APL '02: Proceedings of the 2002 conference on APL, 2002, pp. 190 – 197.

131. Serna, E. (2011). La abstracción como componente crítico de la formación en ciencias computacionales. En Revista: Avances en Sistemas e Informática, vol. 8, núm. 3, diciembre, 2011, pp. 79 – 83.

132. Siegel, S. (1972). Diseño experimental no paramétrico aplicado a las ciencias de la conducta. Edit. Revolucionaria, Cuba.

133. Skiena, S. S. (2008). The algorithm design manual. Second edition. Pub. Springer. ISBN: 978 – 1 – 8 4800 – 069 – 8.

134. Soler, Y., Frías, I., Linares, M. J., Rodríguez, E. A. y Lezcano, M. (2008). Mapa conceptual tipos abstractos de datos y sistema de visualización de programas SVP – SUBC: herramientas eficaces en la formación virtual del ingeniero informático. Congreso Virtual Iberoamericano de Calidad en Educación a Distancia. Disponible en: http://es.scribd.com/doc/21739903/RD14. pp.1 – 13. [Consultado el 10 de Enero 2013].

135. Stroustrup, B. (2000). The C++ Programming Language. Addison – Wesley Pub Co; Tercera edición; ISBN 0 – 201 – 70073 – 5.

136. Syme, D., Granicz, A. y Cisternino, A. (2010). Expert F# 2.0. Apress, Springer – Verlag, New York.

137. Tan, J., Guo, X., Zheng, W. y Zhong, M. (2014). Case – based teaching using the Laboratory Animal System for learning C/C++ programming. En Revista Computers & Education. Volumen 77, Agosto 2014, pp. 39 – 49.

138. Torres, P. (1992). La enseñanza problémica de la matemática del nivel medio general. Tesis en opción al grado científico de Doctor en Ciencias Pedagógicas. Ciudad de la Habana. Cuba.

139. Urrutia, I. y 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. ISSN: 1728 – 6042, RNPS 2017, Vol. 5. Número especial (2009).

140. Van Dalen, B. (1996). Al' Khwarizmi's astronomical tables revisited: analysis of the equation of time. En Annuari de Filologia (Universitat de Barcelona) XIX (1996) B–2. Barcelona: Universitat de Barcelona – Instituto Millás Vallicrosa de Historia de la Ciencia Árabe. pp. 195 – 252.

141. Vargas, A., Pérez, O. L. y Blanco, R. (2014). Desarrollo de la habilidad algoritmizar: una visión desde los estudios de Ciencia, Tecnología y Sociedad. 1ra Conferencia Científica Internacional UCIENCIA 2014.

Universidad de las Ciencias Informáticas. Cuba. Disponible en: https://uciencia.uci.cu/sites/default/files/public/p3499-ponencia-1062_0.pdf. [Consultado el 12 de Junio de 2014].

142. Wang, J., Mendori, T. y Xiong, J. (2014). A language learning support system using course – centered ontology and its evaluation. En Revista Computers & Education. Volumen 78, pp. 278 – 293.
143. Wengrowicz, N. (2014). Teachers' pedagogical change mechanism – Pattern of structural relations between teachers' pedagogical characteristics and teachers' perceptions of transactional distance (TTD) in different teaching environments. En Revista Computers & Education 76 (2014) 190–198.
144. Whimbey, A. y Lochhead, J. (1993). Comprender y Resolver Problemas, Visor Distribuciones. España.
145. Whitfield, A. K. y otros. (2007). Programming, disciplines and methods adopted at Liverpool Hope University. ITALICS Volume 6 Issue 4, October 2007 [ISSN: 1473 – 7507]
146. Wiltrock, R. (1990). Comprensión y representación. MacMillan Publishing Company.
147. Wing, J. M (2006). Computational Thinking. Disponible en: http://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf [Consultado el 15 de Mayo de 2013].
148. Wirth, N. y Hoare, C. A. R. (1966). A contribution to development of ALGOL. Community of the ACM 9 (6), pp. 413 – 432.
149. Yamane, T. (1980). Elementary sampling theory. Editorial Pueblo y Educación. La Habana, Cuba.
150. Yasar, O. (2013). Computational Math, Science and Technology (C-MST). Approach to General Education Courses. En Journal of Computational Science Education. Vol. 4, November, No. 1, pp. 2 – 10.
151. Zamora, L. y Díaz, J. R. (2008). Aplicación del análisis estadístico implicativo al estudio del rendimiento académico de estudiantes de primer año de las carreras de Matemática y Ciencia de la Computación. Cadernos do IME, Série Estatística – Volume 25.
152. Zamora, L., Orús, P. y Díaz, J. R. (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.

153. Zarazaga, F. J. y Alfonso, M. I. (2003). La Ingeniería del Software en el currículo del Ingeniero en Informática. En Novática: Revista de la Asociación de Técnicos de Informática, ISSN 0211 – 2124, No. 161, 2003, pp. 43 – 50. Disponible en: http://dialnet.unirioja.es/servlet/articulo?codigo=312365. [Consultado el 21 de Agosto 2012].

154. Zuse, K. (1993). The Computer – My Life. Berlin / Heidelberg: Springer – Verlag. ISBN 0-387-56453-5. Disponible en: http://es.wikipedia.org/w/index.php?title=KonradZuse&am [Consultado el 5 de Agosto 2011].

ANEXO 1

RESULTADOS EVALUATIVOS OBTENIDOS POR LOS ESTUDIANTES DE LAS CARRERAS DE CIENCIAS COMPUTACIONALES DE LA UNIVERSIDAD DE ORIENTE EN LA ASIGNATURA DE PROGRAMACIÓN

En este primer acercamiento al problema de la investigación se empleó como **técnica** para recopilar información, la **revisión documental**, consistente en el análisis de los resultados evaluativos obtenidos por los estudiantes de las cuatro carreras de ciencias computacionales de la Universidad de Oriente en la asignatura Programación, en los cursos 2003 – 2004 al 2011 – 2012. Así mismo, se usó como **instrumento** una **tabulación** o **matriz de datos**.

Como puede observarse a continuación, el procesamiento de la información obtenida con esa técnica cualitativa se realizó de manera cuantitativa a partir de la estadística descriptiva. El estudio documental realizado permitió en primer lugar una comprensión y una actualización sobre el tema investigado, en este caso el comportamiento de los resultados evaluativos en la mencionada asignatura. Además permitió redescubrir hechos, sugerir problemas, y realizar valoraciones. En resumen contribuyó a fundamentar el propósito de la investigación permitiendo el desarrollo del marco teórico y/o conceptual, que se inscribe en este tipo de investigación fundamentalmente teórica.

En la Tabla 1.1 se muestran los resultados de promoción final de la asignatura de Programación de la carrera Licenciatura en Ciencia de la Computación:

Tabla 1.1: Resultados evaluativos obtenidos por los estudiantes de primer año de la carrera de Licenciatura en Ciencia de la Computación desde el curso 2003 – 2004 al 2011 – 2012 [Fuente: Secretaría docente de la Facultad de Matemática y Computación].

PLAN DE ESTUDIO	CURSO	MATRÍCULA	CANTIDAD APROBADOS	POR CIENTO
Plan C Perfeccionado	2003 – 2004	94	57	60,64
	2004 – 2005	100	45	45,00
	2005 – 2006	85	50	58,82
	2006 – 2007	101	45	44,55
	2007 – 2008	112	46	41,07
				Promedio
				50,01
Plan D	2008 – 2009	113	71	62,83
	2009 – 2010	116	57	49,14
	2010 – 2011	97	50	51,55
	2011 – 2012	48	23	47,92
				Promedio
				52,86

Como puede observarse en la tabla, los resultados no son satisfactorios pues en sólo dos cursos se han alcanzado promociones por encima del 60%, llegándose a obtener como resultado más desfavorable en el período analizado el 41,07%, el cual se manifestó en el curso 2007 – 2008. Los promedios para ambos planes de estudio están en un rango de 50,01 a 52,86 %.

También es destacable el hecho de que no se observe una diferencia significativa entre los resultados obtenidos en cada uno de los planes de estudios analizados, lo que está en correspondencia con el diseño curricular de dichos planes para el caso de la asignatura de Programación, ya que los contenidos no sufrieron cambios importantes.

Del análisis anterior se concluye que los estudiantes de la carrera de Ciencia de la Computación de la Universidad de Oriente, han manifestado **bajos resultados evaluativos en la asignatura de Programación.**

En la Tabla 1.2 se muestran los resultados de la promoción final en la asignatura de Programación de la carrera Ingeniería Informática:

Tabla 1.2: Resultados evaluativos obtenidos por los estudiantes de primer año de la carrera de Ingeniería Informática desde el curso 2003 – 2004 al 2011 – 2012 [Fuente: Secretaría docente de la Facultad de Ingeniería Eléctrica].

PLAN DE ESTUDIO	CURSO	MATRÍCULA	CANTIDAD APROBADOS	POR CIENTO
Plan C Perfeccionado	2003 – 2004	71	53	74,65
	2004 – 2005	42	38	90,48
	2005 – 2006	87	58	66,67
	2006 – 2007	96	68	70,83
	2007 – 2008	57	49	85,96
				Promedio
				77,92
Plan D	2008 – 2009	61	47	77,05
	2009 – 2010	43	13	30,23
	2010 – 2011	67	24	35,82
	2011 – 2012	53	19	35,84
				Promedio
				44,74

Como puede observarse, los resultados de los cursos 2003 – 2004 al 2007 – 2008 se pueden catalogar de satisfactorios, con un porciento de aprobados en cada curso de más del 70, a excepción del curso 2005 – 2006 en que desciende al 66 %. Si bien no se puede decir que estos son los porcientos deseados para una carrera cuyo objetivo principal es la programación, sí se mantuvo estable durante los cinco cursos mencionados.

En cuanto a los cursos 2008 – 2009 al 2010 – 2011, pertenecientes al Plan de Estudios D, se aprecia que sólo en el primero se alcanzaron promociones por encima del 77 %, siendo menor que la media general de los

cinco cursos anteriores. En los siguientes cursos las promociones en la asignatura estuvieron por debajo del 36%, llegándose a obtener una promoción mínima del 30,23 % en el curso 2009 – 2010, lo cual evidencia la existencia de insuficiencias en el proceso de enseñanza – aprendizaje de la asignatura.

Es destacable el hecho de que se observa una diferencia significativa entre los resultados obtenidos en cada uno de los planes de estudios analizados, lo que está en correspondencia con el diseño curricular de dichos planes para el caso de esta asignatura de Programación, ya que aunque los contenidos de la misma no sufrieron cambios importantes, la distribución de éstos sí se modificó.

Del análisis anterior se concluye que los estudiantes de la carrera de Ingeniería Informática de la Universidad de Oriente, al igual que los de Licenciatura en Ciencia de la Computación, han manifestado **bajos resultados evaluativos en la asignatura de Programación.** Cabe destacar que en el caso de estas dos carreras no se realizó un análisis de la calidad de la promoción debido a los bajos resultados alcanzados, que en sí mismos dan cuenta de las deficiencias globales del proceso.

En la Tabla 1.3 de este anexo se muestran los resultados de la promoción final en la asignatura de Programación para la carrera de Ingeniería en Telecomunicaciones y Electrónica. Además se incluyen los resultados del análisis de la calidad de dicha promoción, dado que la misma en esta carrera es superior a la de las dos anteriores, manteniéndose por encima del 80%.

Tabla 1.3: Resultados evaluativos obtenidos por los estudiantes de primer año de la carrera de Ingeniería en Telecomunicaciones y Electrónica desde el curso 2003 – 2004 al 2011 – 2012 [Fuente: Secretaría docente de la Facultad de Ingeniería Eléctrica].

PLAN DE ESTUDIO	CURSO	MATRÍCULA	CANTIDAD APROBADOS	POR CIENTO	NOTAS DE 3 PUNTOS	POR CIENTO
Plan C Perfeccionado	2003 – 2004	68	66	97,10	38	57,58
	2004 – 2005	75	69	92,00	36	52,17
	2005 – 2006	88	78	88,64	40	51,28
	2006 – 2007	100	99	99,00	55	55,56
	2007 – 2008	96	59	61,46	41	69,49
				Promedio		**Promedio**
				87,64		**57,22**
Plan D	2008 – 2009	104	90	86,54	50	55,56
	2009 – 2010	102	85	83,33	52	61,18
	2010 – 2011	112	92	82,14	67	72,83
	2011 – 2012	84	64	76,19	49	76,56
				Promedio		**Promedio**
				82,05		**66,53**

Como puede observarse en la tabla que se presenta, los resultados en cuanto a promoción final son satisfactorios pues en sólo dos cursos se han alcanzado promociones por debajo del 80 %, sin embargo no puede decirse lo mismo de la calidad de la promoción, ya que durante el plan C perfeccionado el promedio de

estudiantes con notas de 3 puntos fue de 57,22% y con la implantación del plan D aumentó al 66,53%, llegándose a obtener un 76,56% en el curso 2011 – 2012.

También es destacable el hecho de que se observe una diferencia significativa en los resultados obtenidos para cada uno de los planes de estudios analizados, evidenciándose un descenso sistemático en la promoción final y un aumento en la cantidad de estudiantes que promueven la asignatura de Programación con 3 puntos.

Del análisis anterior se concluye que los estudiantes de la carrera de Ingeniería en Telecomunicaciones y Electrónica de la Universidad de Oriente, han manifestado **baja calidad en los resultados evaluativos en la asignatura de Programación**, los que se han venido agravando en los últimos cuatro cursos, reflejándose principalmente en la promoción y en la calidad de la misma.

En la Tabla 1.4 se muestran los resultados de la promoción final en la asignatura de Programación para la carrera Ingeniería en Automática y los resultados del análisis de la calidad de dicha promoción:

Tabla 1.4: Resultados evaluativos obtenidos por los estudiantes de primer año de la carrera de Ingeniería en Automática desde el curso 2003 – 2004 al 2011 – 2012 [Fuente: Secretaría docente de la Facultad de Ingeniería Eléctrica].

PLAN DE ESTUDIO	CURSO	MATRÍCULA	CANTIDAD APROBADOS	POR CIENTO	NOTAS DE 3 PUNTOS	POR CIENTO
Plan C Perfeccionado	2003 – 2004	51	43	84,31	19	44,18
	2004 – 2005	58	54	93,10	42	77,78
	2005 – 2006	68	57	83,82	43	75,44
	2006 – 2007	68	68	100,00	29	42,65
	2007 – 2008	60	60	100,00	27	45,00
				Promedio		**Promedio**
				92,25		**57,01**
Plan D	2008 – 2009	50	45	90,00	18	40,00
	2009 – 2010	66	63	95,45	32	50,79
	2010 – 2011	66	44	66,67	36	81,81
	2011 – 2012	63	59	93,65	42	71,19
				Promedio		**Promedio**
				86,44		**60,95**

Como puede observarse los resultados en cuanto a promoción final son satisfactorios, pues en un sólo curso se alcanzó una promoción por debajo del 80 %. Sin embargo, no se puede decir lo mismo de la calidad de la promoción, la que durante el Plan de Estudio C perfeccionado no fue buena, con un 57,01% de calificaciones de 3 puntos como promedio. Con la implantación del Plan de Estudio D aumentó dicho promedio a un 60,95%, llegándose a obtener un 81,81% en el curso 2010 – 2011.

También es destacable el hecho de que se observen diferencias en los resultados obtenidos para cada uno de los planes de estudio analizados, evidenciándose un aumento en la cantidad de estudiantes que promueven la asignatura de Programación con 3 puntos en el Plan de Estudio D.

Del análisis anterior se concluye que los estudiantes de la carrera de Ingeniería en Automática de la Universidad de Oriente, han manifestado **baja calidad en los resultados evaluativos en la asignatura de Programación**, que se han venido agravando en los últimos cursos, reflejándose principalmente en la promoción final y en la calidad de la misma.

ANEXO 2

GUÍA PARA LA PRIMERA ENTREVISTA A PROFESORES DE PROGRAMACIÓN

Esta segunda **técnica** consistió en una **entrevista no estructurada** e **individual** y se usó como **instrumento** una **guía de entrevista**. La misma se realizó con el **objetivo** de determinar las principales insuficiencias que se presentan en la resolución de problemas de programación computacional y sus causas fundamentales. Esta se aplicó a 24 profesores, los que constituyen el 96% de los que imparten Programación en las carreras de ciencias computacionales de la Universidad de Oriente, tomando como indicadores:

1. Valoración sobre los resultados docentes que históricamente se han obtenido en la asignatura de Programación.
2. Principales deficiencias que presentan los estudiantes al programar.
3. Métodos y medios de enseñanza más utilizados por el profesor para facilitar el aprendizaje de la Programación.
4. Formas de enseñanza y tipos de evaluaciones más usadas en la asignatura.

Metodología de desarrollo de la primera entrevista: Las entrevistas se realizaron de manera presencial y fueron grabadas. En general, la duración fue de 30 a 40 minutos. Una vez finalizado el trabajo de campo (transcripciones de las grabaciones en audio y de las notas tomadas durante las entrevistas) se procedió al análisis del contenido de los datos textuales. Se clasificó la información recogida siguiendo los aspectos previstos en la guía y se realizaron recuentos simples de respuestas similares, generadas respecto a las cuestiones planteadas en las entrevistas. De este modo, se obtuvo un guión que permitió desarrollar categorías analíticas y explicaciones teóricas.

Esta estructura se aplicó de forma sistemática a todas las transcripciones, para permitir codificar los datos de las entrevistas según sus categorías, de modo que pudieran analizarse los datos reorganizados. Simultáneamente, se seleccionaron los comentarios claves, susceptibles de ser utilizados posteriormente como citas textuales, los que se muestran en el Anexo 3 entrecomillados.

Durante todo el proceso analítico antes mencionado se emplearon los **métodos** teóricos de análisis – síntesis e inducción – deducción, logrando expresar, ordenar, describir e interpretar los datos mediante conceptos y razonamientos, realizándose valoraciones y arribando a conclusiones que permitieron plantear las principales insuficiencias que generaron el problema de la investigación y sus posibles causas, algunas expresadas por los encuestados y otras obtenidas como producto de la interpretación y construcción del conocimiento del propio investigador.

ANEXO 3

PROCESAMIENTO DE LOS DATOS A PARTIR DE LA PRIMERA ENTREVISTA

A PROFESORES DE PROGRAMACIÓN

P1. Valoración sobre los resultados docentes que históricamente se han obtenido en la asignatura de Programación.

- ***Insuficiencias***

La generalidad de los especialistas entrevistados considera que los resultados docentes obtenidos en la asignatura en los últimos 10 años no son buenos, lo que se refleja en la cantidad de estudiantes que abandonan la carrera en el primer año, desaprueban o alcanzan calificaciones de 3 puntos.

« (…) en Ciencia de la Computación aprueba alrededor del 50 % en los últimos 10 años. Aunque se debe analizar que los grupos actuales tienen gran cantidad de alumnos y anteriormente no era así (…) » (E-1).

« (…) con el nuevo Plan D de Informática por lo general aprueba menos del 50 % (…) » (E-12).

- ***Causas:***

« (…) la calidad de los alumnos en cuanto a preparación en la enseñanza precedente era mejor (…) » (E-15).

« (…) no hay una buena preparación en matemática en la enseñanza anterior (…) » (E-5).

« (…) los estudiantes no tienen una buena base matemática al entrar en la universidad, lo que dificulta el proceso de aprendizaje (…) » (E-3).

Conclusiones del ítem: Todos los especialistas entrevistados coinciden en que los resultados docentes en la asignatura en los últimos diez años están alrededor del 50%. Lo que confirma la necesidad de incidir en este proceso con el fin de perfeccionarlo para elevar el por ciento de aprobados.

P2. Principales deficiencias que presentan los estudiantes al programar.

- ***Insuficiencias***

En este ítem, las opiniones más relevantes de los especialistas se dirigieron a señalar insuficiencias relativas a las habilidades para modelar matemáticamente una situación problémica, al respecto consideran que:

« (…) no saben interpretar ni representar matemáticamente un problema. No tienen creatividad en las soluciones. (…) no tienen independencia a la hora de estudiar (…) » (E-4).

« La mayoría de los estudiantes presentan dificultades al analizar un problema y modelarlo matemáticamente » (E-7).

« Los estudiantes que presentan los mayores problemas para algoritmizar son aquellos que llegan a la universidad con un insuficiente dominio de la Matemática (…) » (E-5).

« Faltan elementos del pensamiento abstracto que permiten que los estudiantes lleven la tarea a algoritmos para obtener la solución » (E-10).

- ***Causas:***

« (…) mala preparación en la enseñanza media, en las asignaturas de ciencias exactas, la que no facilita el desarrollo de la capacidad de análisis y resolución de problemas (…) » (E-22).

« (…) no tienen una lógica adecuada de programación.» (E-6).

« (…) no se forma en el estudiante un pensamiento lógico de cómo resolver problemas.» (E-4).

Conclusiones del ítem: Estos especialistas reconocen como aspectos esenciales, deficientes en el proceso en estudio, la comprensión, interpretación, representación, análisis y modelación de una situación problémica, unido a la ausencia de una lógica de programación formada.

P3. Métodos de enseñanza más utilizados por el profesor para facilitar el aprendizaje de la Programación.

En este aspecto las valoraciones de los especialistas se inclinaron a favor del uso de ejemplos como recursos principales para enseñar programación, sobresaliendo las siguientes opiniones:

« (…) elaboración del algoritmos a través de ejemplos y uso de analogías para enseñar los conceptos principales (…) » (E-6).

« Uso de ejemplos para enseñar los conceptos: se pintan en la pizarra las estructuras para que les sea más fácil abstraerse, luego se le presenta un algoritmo en pseudocódigo que se corresponde con el dibujo y se le orienta su posterior implementación en un lenguaje (…) » (E-24).

« (…) se emplea el método de elaboración conjunta, a partir de proponer un ejercicio y hallar su solución y trabajarlo entre todos (…) » (E-8).

« (…) se deben usar ejemplos para que los estudiantes se apropien de los conceptos enseñados en la conferencia, pues hasta que los estudiantes no ven ejemplos concretos, no son capaces de abstraerse (…) » (E-20).

« Se resuelven problemas a través de la algoritmización, explicándole al estudiante los métodos paso a paso y luego ejemplificándola, sin embargo el tiempo es escaso y no se puede hacer mucho énfasis en la algoritmización (…) » (E-9).

Conclusiones del ítem: La mayoría de los especialistas encuestados priorizan el uso de ejemplos y analogías como métodos principales para facilitar el aprendizaje de la Programación. Otros introducen el método de la elaboración conjunta. Aquí es preciso enfatizar la importancia que éstos le confieren a la algoritmización, reconociendo que a pesar de ello en la práctica no dedican suficiente tiempo a trabajarla.

P4. Formas de enseñanza y tipos de evaluaciones más usadas en la asignatura.

En este punto los criterios de los especialistas se encuentran divididos entre los que consideran que se deben aumentar las horas de laboratorio y los que valoran un aumento de las clases prácticas; sobresalen en estos aspectos las siguientes opiniones:

« La asignatura tiene predominio de las clases prácticas y los laboratorios, realizándose dos trabajos de control y un trabajo extra-clase, de manera individual; además de evaluaciones sistemáticas y un examen final (…) » (E-8).

« (…) considero que además de las clases prácticas se podrían aumentar las horas de laboratorio, para potenciar la ejercitación de los alumnos en el uso del Builder C++ y en el diseño de algoritmos (…) » (E-3).

« (…) se podrían aumentar las horas de clases prácticas, pero encaminadas a diseñar algoritmos usando pseudocódigos o diagramas de flujo, como se hacía con FORTRAN, sin usar ningún lenguaje específico (…) » (E-10).

« (…) se debería usar algún mediador didáctico en la computadora, como por ejemplo un software que enseñara los pasos para algoritmizar, mostrando lo que sucede en la computadora internamente a través de ejemplos con distintos niveles de complejidad. Esto se hace necesario, pues con la complejidad que presentan los actuales sistemas para programar, tales como el Builder C++, Jbuilder, entre otros, a los estudiantes se les hace muy complejo su uso sin tener una idea de lo que sucede realmente cuando se utiliza, por ejemplo, una estructura computacional, o se declara una variable (…) » (E-5).

Conclusiones del ítem: Se aprecia que en este punto los especialistas coinciden en que, aun cuando los tipos de actividades están distribuidas de manera balanceada, se podrían aumentar las frecuencias de clases prácticas y laboratorios con el objetivo de potenciar en el estudiante el desarrollo de habilidades para la programación.

ANEXO 4

ENCUESTA A ESTUDIANTES DE 2do AÑO DE LAS CARRERAS DE LICENCIATURA EN CIENCIA DE LA COMPUTACIÓN Y DE INGENIERÍA INFORMÁTICA DE LA UNIVERSIDAD DE ORIENTE

Estimado estudiante:

La presente encuesta forma parte de un estudio que tiene el objetivo de perfeccionar la enseñanza de la asignatura Programación. Le solicitamos que, por favor, lea de manera cuidadosa la información que se le solicita y responda con sinceridad todas las preguntas, empleando para ello las siguientes categorías:

1	2	3	4	5
Totalmente en desacuerdo	En desacuerdo	Ni de acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo

Le hacemos llegar nuestro agradecimiento por su valiosa contribución. Muchas Gracias.

Cuestionario

No	AFIRMACIONES	1	2	3	4	5
1	Replanteo los problemas con mis propias palabras.					
2	No compruebo la solución de cada problema.					
3	Leo el planteamiento del problema varias veces antes de tratar de resolverlo.					
4	No utilizo gráficos, tablas, ecuaciones y funciones para representar el problema que estoy solucionando.					
5	Distingo la información relevante de la irrelevante en cada problema antes de solucionarlo.					
6	No comienzo a resolverlo hasta no estar seguro de que he interpretado de manera clara y precisa cada uno de sus elementos y tengo una visión integrada de ellos.					
7	Exploro diferentes estructuras computacionales (for, if, then, while, etc.) antes de diseñar un algoritmo de solución para el problema.					
8	Para dar solución a un problema diseño el programa sin tener en cuenta un orden lógico para utilizar las estructuras computacionales (for, if, then, while, etc.)					
9	Estimo la respuesta final después de concebir una vía para alcanzar la solución.					
10	Después de hallar la solución no pruebo otros datos para ejecutar (correr) el programa.					
11	Reflexiono sobre el método o los métodos que utilizo para solucionar un problema, después de solucionado.					
12	No diseño el algoritmo antes de implementar en un lenguaje particular.					
13	Utilizo pseudocódigo para diseñar el algoritmo antes de implementarlo.					
14	Al resolver un problema comienzo a implementarlo en un lenguaje sin utilizar pseudocódigos.					
15	Dominar la sintaxis o reglas de un lenguaje es más importante que saber algoritmizar.					
16	Si un programa cumple con la sintaxis de un lenguaje y se ejecuta correctamente, entonces el resultado esperado cumple con el objetivo para el que fue implementado.					

ANEXO 5

ORGANIZACIÓN TEÓRICA DE LA INFORMACIÓN PARA EL PROCESAMIENTO DE LA ENCUESTA A ESTUDIANTES DE 2do AÑO DE LAS CARRERAS DE LICENCIATURA EN CIENCIA DE LA COMPUTACIÓN Y DE INGENIERÍA INFORMÁTICA DE LA UNIVERSIDAD DE ORIENTE

Para el procesamiento de la información se organizaron los datos en una tabla bidimensional o de doble entrada (Gorina, A, 2010; Gorina, A., Alonso, I. y Zamora, L., 2010):

Tabla 5.1: Tabla bidimensional para representar los datos por ítems y por estudiantes. [Fuente: Elaboración propia].

Ítems / Estudiantes	1	2	3	...	J	...	n
1	C_{11}	C_{12}	C_{13}	...	C_{1j}	...	C_{1n}
2	C_{21}	C_{22}	C_{23}	...	C_{2j}	...	C_{2n}
3	C_{31}	C_{32}	C_{33}	...	C_{3j}	...	C_{3n}
…	…	…	…	...	…	...	…
i	C_{i1}	C_{i2}	C_{i3}	...	C_{ij}	...	C_{in}
…	…	…	…	...	…	...	…
m	C_{m1}	C_{m2}	C_{m3}	...	C_{mj}	...	C_{mn}

Tabla en la que:

m: cantidad de estudiantes;

n: cantidad de ítems;

mj: cantidad de estudiantes que evalúan el ítem j (j = 1…n y mj ≤ m);

Cij: evaluación en puntos del ítem j por el estudiante i.

A partir de la tabla anterior se obtuvieron las siguientes estimaciones:

- *Criterio generalizado para un ítem dado*, que se expresa a partir de la media aritmética de los puntajes para el ítem *j* (media de la columna *j*): $\overline{C_j} = \frac{\sum_{i=1}^{mj} C_{ij}}{mj}$
- *Grado de concordancia de los estudiantes para un ítem dado,* que se expresa a partir de la varianza (σ_j^2), la desviación típica (σ_j) o el coeficiente de variación (v_j) de los puntajes obtenidos en el ítem *j*:

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

Cabe destacar que se le prestó mayor atención al coeficiente de variación, el cual es la desviación estándar expresada como un por ciento del promedio, por lo que proporciona una medida de la magnitud de la variación relativa al tamaño de la cantidad que se mide. Por lo tanto, a mayor valor de v_j menor será el grado de concordancia de los m estudiantes con relación al ítem j.

Se asumió como valor plausible para el umbral o punto de corte de v_j, para aceptar una adecuada concordancia de los estudiantes en el ítem j, al valor 0,25; que representa un cuarto de v_j (Martínez, J. A. y Martínez, L., 2008).

Aplicación del muestreo probabilístico por conglomerados bietápico

Notación:

N: número de conglomerados en la población.

n: número de conglomerados en la muestra.

m_i : número de unidades elementales o primarias en el i-ésimo conglomerado.

$M = \sum_{i=1}^{N} m_i$: total de elementos en la población.

$\overline{M} = \frac{M}{N}$: tamaño promedio de los conglomerados en la población.

y_i : total del conglomerado i-ésimo.

$\overline{m} = \frac{\sum_{i=1}^{n} m_i}{n}$: tamaño promedio del conglomerado en la muestra.

Estimación de la media poblacional

Se utilizó el siguiente estimador para estimar la Media Poblacional:

$\hat{\mu} = \overline{y} = \frac{\sum_{i=1}^{n} \hat{y}_i}{\sum_{i=1}^{n} m_i}$, donde $\hat{y}_i$ es el estimador del total del conglomerado i-ésimo.

Este estimador de la media tiene la *forma de un estimador de razón*, por lo tanto, la varianza de la media tiene la forma de la varianza del estimador de razón:

$$\hat{V}\left(\overline{y}\right) = \left(\frac{N-n}{Nn\overline{M}^2}\right)\frac{\sum_{i=1}^{n}\left(\hat{y}_i - \overline{y}m_i\right)^2}{n-1}$$

Si se desconoce el total de elementos en la población M, entonces, $\overline{M}$ puede ser estimado con

$$\overline{m} = \frac{\sum_{i=1}^{n} m_i}{n}.$$

El límite para el *error de estimación* es: $e = B = t_{1-\frac{\alpha}{2}} \sqrt{\hat{V}\left(\bar{y}\right)} = t_{1-\frac{\alpha}{2}} \sqrt{\left(\frac{N-n}{Nn\overline{M}^2}\right) \frac{\sum_{i=1}^{n} \left(\hat{y}_i - \bar{y} m_i\right)^2}{n-1}}$

Los *límites de confianza* son: $\bar{y} \pm e$.

Estimación del total poblacional

El total poblacional τ puede ser determinado por $M\mu$, porque *M* denota el total de elementos en la población. Por lo tanto, así como en el muestreo aleatorio simple, el puede ser estimado por:

$$\hat{\tau} = M\bar{y} = M \frac{\sum_{i=1}^{n} \hat{y}_i}{\sum_{i=1}^{n} m_i}$$

La varianza estimada de $\hat{\tau} = M\bar{y}$: $\hat{V}\left(\hat{\tau}\right) = \hat{V}\left(M\bar{y}\right) = M^2 \hat{V}\left(\bar{y}\right) = N^2 \left(\frac{N-n}{Nn}\right) \frac{\sum_{i=1}^{n} \left(\hat{y}_i - \bar{y} m_i\right)^2}{n-1}$

El límite para el error de estimación es: $e = B = t_{1-\frac{\alpha}{2}} \sqrt{\hat{V}\left(\hat{\tau}\right)} = t_{1-\frac{\alpha}{2}} \sqrt{N^2 \left(\frac{N-n}{Nn}\right) \frac{\sum_{i=1}^{n} \left(\hat{y}_i - \bar{y} m_i\right)^2}{n-1}}$

Por lo que *e* = 0,11 siendo el error máximo admisible o precisión mínima tolerable al estimador de la media poblacional utilizado.

Selección del Tamaño de Muestra

Por definición el error de estimación es:

$e = B = t_{1-\frac{\alpha}{2}} \sqrt{V\left(\bar{y}\right)} = t_{1-\frac{\alpha}{2}} \sqrt{\left(\frac{N-n}{Nn\overline{M}^2}\right)\sigma_c^2}$, donde $\sigma_c^2 = \frac{\sum_{i=1}^{N} \left(\hat{y}_i - \mu m_i\right)^2}{N-1}$

y $V\left(\bar{y}\right) = \left(\frac{N-n}{Nn\overline{M}^2}\right)\sigma_c^2$ es la varianza poblacional y $\hat{V}\left(\bar{y}\right) = \left(\frac{N-n}{Nn\overline{M}^2}\right)S_c^2$ es la varianza estimada.

Al despejar de la fórmula del error de estimación el valor de *n*, se obtiene que el tamaño de muestra es:

$n = \frac{N\sigma_c^2}{ND + \sigma_c^2}$, denotando por $D = \left(\frac{e}{t_{1-\frac{\alpha}{2}}}\right)^2 \overline{M}^2$.

ANEXO 6

PROCESAMIENTO DE LA ENCUESTA A ESTUDIANTES DEL 2do AÑO DE LAS CARRERAS DE LICENCIATURA EN CIENCIA DE LA COMPUTACIÓN Y DE INGENIERÍA INFORMÁTICA DE LA UNIVERSIDAD DE ORIENTE

En la Tabla 6.1 se sintetizan los datos obtenidos mediante la encuesta a los 18 estudiantes de la carrera de Licenciatura en Ciencia de la Computación, con relación a cada uno de los 16 ítems. Además se muestran las medidas estadísticas que se tuvieron en cuenta para caracterizar el criterio generalizado y el nivel de concordancia para cada ítem de la encuesta.

Tabla 6.1: Valoraciones de los estudiantes para cada ítem de la encuesta. Media aritmética de los criterios de los estudiantes y medidas estadísticas de dispersión para valorar el nivel de concordancia por ítem. [Fuente: Elaboración propia].

ÍTEM / EST	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
E-1	4	5	5	5	5	1	5	5	4	5	5	5	4	4	5	4
E-2	4	4	3	3	4	2	4	3	4	4	4	2	4	4	2	2
E-3	4	4	5	4	4	1	3	4	4	5	4	1	4	3	4	2
E-4	5	4	5	3	4	1	4	3	4	5	5	3	3	2	3	1
E-5	3	3	5	4	3	2	4	4	4	5	3	2	4	1	3	2
E-6	4	4	5	2	1	1	5	3	3	5	1	2	4	2	3	1
E-7	4	4	5	5	4	2	4	5	4	4	5	5	1	4	4	2
E-8	1	1	3	3	4	4	2	5	3	5	2	2	3	2	3	3
E-9	2	4	4	5	5	2	5	3	2	5	5	3	2	1	3	3
E-10	1	2	1	3	2	4	2	1	1	3	3	2	2	2	4	3
E-11	3	3	5	3	4	2	5	4	5	4	4	4	1	4	3	2
E-12	4	2	4	4	5	4	2	4	2	2	4	1	2	1	4	5
E-13	4	2	4	4	4	2	4	4	4	4	4	4	3	3	3	2
E-14	4	5	5	4	3	1	3	3	5	5	4	2	3	2	4	4
E-15	4	4	4	4	3	2	4	5	5	5	4	4	4	3	3	3
E-16	3	5	5	3	4	1	3	5	5	5	3	1	2	4	3	1
E-17	4	4	5	4	4	1	3	4	4	5	4	5	4	3	4	2
E-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
Máx P.	5	5	5	5	5	4	5	5	5	5	5	5	5	4	5	5
Mín P.	1	1	1	2	1	1	1	2	1	2	1	1	1	1	2	1

Donde:

Puntajes (+): la cantidad de valoraciones realizadas con puntajes 4 ó 5.

Puntajes (-): la cantidad de valoraciones realizadas con puntajes 1 ó 2.

Puntajes (+/-): la cantidad de valoraciones realizadas con puntaje 3.

Máximo P, representa el máximo valor de los puntajes y Mínimo P, representa el mínimo valor de los puntajes.

En la Tabla 6.2 se exponen los datos obtenidos mediante la encuesta a los 23 estudiantes de la carrera de Ingeniería Informática, con relación a cada uno de los 16 ítems. Además de las medidas estadísticas que se tuvieron en cuenta para caracterizar el criterio generalizado y el nivel de concordancia para cada ítem.

Tabla 6.2: Valoraciones de los estudiantes para cada ítem de la encuesta. Media aritmética de los criterios de los estudiantes y medidas estadísticas de dispersión para valorar el nivel de concordancia por ítem. [Fuente: Elaboración propia].

ÍTEM / EST	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
E-1	4	5	5	1	4	2	5	5	5	3	3	3	1	5	3	2
E-2	2	2	4	4	4	2	4	4	2	4	4	2	2	4	4	2
E-3	4	5	4	4	3	1	3	4	2	5	4	4	4	3	4	3
E-4	4	5	4	3	5	2	3	4	4	5	3	1	4	2	4	5
E-5	5	4	4	4	5	2	4	4	4	5	4	4	4	4	2	2
E-6	5	3	5	4	4	2	2	3	3	5	5	3	3	4	4	2
E-7	3	4	5	5	3	3	2	4	3	4	4	1	4	5	3	3
E-8	4	4	5	5	4	2	4	4	4	3	4	2	2	1	2	1
E-9	4	4	4	3	3	3	4	3	3	5	5	3	2	1	5	5
E-10	4	4	5	5	4	2	4	4	4	4	4	2	2	4	4	3
E-11	5	5	5	2	4	1	5	5	2	5	5	5	5	1	1	1
E-12	4	4	4	3	4	3	2	4	1	5	3	5	4	2	5	5
E-13	5	5	5	4	4	4	2	5	4	2	4	1	2	3	2	1
E-14	4	5	5	4	4	3	4	3	3	5	5	3	3	2	4	2
E-15	5	4	5	3	4	2	4	3	3	4	4	2	3	3	3	2
E-16	5	5	4	4	4	2	5	5	5	3	1	1	1	1	5	1
E-17	4	4	4	3	3	3	3	3	4	5	3	4	4	4	3	2
E-18	2	4	4	3	4	1	5	5	4	5	5	4	3	4	4	2
E-19	4	4	4	2	5	2	4	5	4	5	3	1	3	1	3	5
E-20	5	5	5	5	5	1	4	4	4	4	4	3	4	3	2	3
E-21	3	3	5	3	5	2	5	4	3	3	4	2	4	2	4	2
E-22	3	4	5	4	3	2	5	5	4	5	2	2	1	2	5	2
E-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
Máx P.	5	5	5	5	5	4	5	5	5	5	5	5	5	5	5	5
Mín P.	2	2	4	1	3	1	3	2	1	2	1	1	1	1	1	1

Consecuentemente, para el procesamiento de la información de la citada encuesta se estructuraron las cuatro clases que se muestran a continuación.

Clase A: $\overline{C_j} \in [1, 2)$ Zona muy desfavorable

Clase B: $\overline{C_j} \in [2, 3)$ Zona desfavorable.

Clase C: $\overline{C_j} \in [3, 4)$ Zona favorable.

Clase D: $\overline{C_j} \in [4, 5]$ Zona muy favorable.

Ahora bien, a nivel poblacional se pudieron realizar las inferencias que se muestran en la Tabla 6.3.

Tabla 6.3: Inferencias a la población de estudiantes de las carreras de ciencias computacionales de la Universidad de Oriente, a partir de un muestreo por conglomerados bietápico.

Indicadores	Media	Varianza	C.V.	Error	Límite Inferior	Límite. Superior	Interpretación cualitativa
I-1	3,827	0,014	0,031	0,234	[3,593	4,061]	Favorable, con rasgos muy favorable
I-2	3,976	0,017	0,033	0,260	[3,716	4,237]	Favorable, con rasgos muy favorable
I-3	4,522	0,002	0,011	0,100	[4,422	4,621]	Muy 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, con rasgos muy favorable
I-6	2,062	0,001	0,016	0,065	[1,998	2,127]	**Desfavorable, con rasgos muy desfavorable**
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]	Muy 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]	Muy favorable
I-11	3,900	0,001	0,008	0,062	[3,838	3,962]	Favorable
I-12	2,856	0,004	0,021	0,119	[2,737	2,975]	**Desfavorable**
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]	**Desfavorable**
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]	**Desfavorable**
Variable operativa	**3,540**	**0,001**	**0,008**	**0,060**	**[3,480**	**3,600]**	**FAVORABLE**

Así mismo, se estimó que la desviación promedio de 3,540 para los datos poblacionales es de 0,001 unidades de la escala. El error absoluto cometido fue de 0,060 (en ambos casos menores que el fijado antes de realizar el muestreo). De manera que, con una confiabilidad del 95%, se puede afirmar que la media poblacional estimada está localizada en el intervalo de confianza [3,480; 3,600]. Por lo tanto, las **inferencias a la población** de estudiantes de las carreras de ciencias computacionales de la Universidad de Oriente, resulto ser **favorable**.

ANEXO 7

GUÍA PARA LA SEGUNDA ENTREVISTA A PROFESORES DE PROGRAMACIÓN

Al igual que en la primera entrevista, esta otra se diseñó de manera **no estructurada** e **individual** y se usó como instrumento una **guía de entrevista.** Su **objetivo** fue profundizar en el objeto y el campo definidos en la investigación. La misma se aplicó a 20 profesores, los que constituyen el 80% de los que imparten Programación en las carreras de Licenciatura en Ciencia de la Computación, Ingeniería Informática, Ingeniería en Telecomunicaciones y Electrónica e Ingeniería en Automática de la UO, tomando como indicadores:

1. Momento del proceso de resolución de problemas de programación computacional en el que se confrontan las mayores dificultades:
   - Análisis e interpretación de la situación problémica.
   - Elaboración del programa.
   - Ejecución y validación del programa.
   - Interpretación de los resultados del programa.
2. Otros aspectos que considere relevantes para el proceso de enseñanza – aprendizaje de la programación.
3. ¿Cómo le gustaría a usted que le enseñaran programación?

Aquí se empleó la misma metodología de desarrollo que en la primera entrevista, de modo que pudieran analizarse los datos reorganizados como se muestra en el anexo 8.

Durante todo el proceso analítico antes mencionado se emplearon los métodos teóricos de análisis – síntesis e inducción – deducción logrando expresar, ordenar, describir e interpretar los datos mediante conceptos y razonamientos, realizándose valoraciones y arribando a conclusiones, que permitieron caracterizar el estado actual del proceso bajo estudio, algunas expresadas por los encuestados y otras obtenidas como producto de la interpretación y construcción del conocimiento del propio investigador.

ANEXO 8

PROCESAMIENTO DE LOS DATOS OBTENIDOS A PARTIR DE LA SEGUNDA ENTREVISTA A PROFESORES DE PROGRAMACIÓN

1. Momento del proceso de programación en el que se confrontan las mayores dificultades (análisis e interpretación de la situación problémica, elaboración del programa, ejecución y validación del programa, interpretación de los resultados del programa).

En este ítem, las opiniones más relevantes de los profesores se pueden clasificar según dos insuficiencias relativas a: la aplicación de un lenguaje de alto nivel para la programación sin concebir y emplear un algoritmo previo y las habilidades para algoritmizar. En el caso de la primera las opiniones más frecuentes fueron:

« (…) No importa el lenguaje en el que se implemente el algoritmo diseñado en la etapa de elaboración del programa, pues en ocasiones la sintaxis de C++ atrasa a los estudiantes con aspectos que no son importantes (…)» (E-2).

« (…) el lenguaje que se use para implementar y ejecutar el programa no es importante pues el algoritmo se puede adaptar a cualesquiera de los existentes (…) » (E-10).

« (…) en la programación no es relevante la parte sintáctica que se lleva a cabo en la ejecución, pues de eso se encargan los compiladores (…) no importa el lenguaje de programación sino el algoritmo desarrollado en la etapa de elaboración de programa (…) hacer el programa es mecánico, no así el diseño (…)» (E-1).

Los profesores coinciden en que el uso de un lenguaje para enseñar de manera directa a programar no es adecuado, pues el estudiante pierde demasiado tiempo aprendiendo la sintaxis y no dedica tiempo a crear habilidades de algoritmización, que a fin de cuentas es lo que le permitirá resolver un problema correctamente. También precisan que cuando se tiene el algoritmo diseñado, su traducción a un lenguaje específico es bastante sencilla, y si se cuenta con compiladores que señalen cualquier error sintáctico, el problema se reduce a que el algoritmo diseñado cumpla con la intencionalidad deseada. Reconocen que no es en el momento de ejecución y validación del programa, en el que los estudiantes confrontan las mayores dificultades, añadiendo que el diseño del algoritmo es lo más importante.

En el segundo caso, relativo a las insuficiencias en las habilidades para algoritmizar, las opiniones más importantes fueron:

« (…) los estudiantes presentan dificultades para algoritmizar, pues pasan trabajo para encontrar la solución matemática, debido a que no tienen una buena formación Matemática y le faltan elementos del pensamiento abstracto que le permitan llevar la tarea a algoritmos, para obtener la solución del problema (…)» (E-1).

« (…) la mayoría no saben algoritmizar, presentando problemas en la modelación matemática. Los estudiantes no corren los algoritmos a mano para verificar su solución, es decir, no hacen una verificación semántica, presentan problemas con las sintaxis del lenguaje (…)» (E-15).

« (…) muchos no saben algoritmizar, y presentan serios problemas a la hora de implementar en un lenguaje. No saben analizar e interpretar un problema (…) no saben diseñar el algoritmo correctamente (…) » (E-3).

« (…) se debe poner énfasis en la comprensión y modelación matemática del problema, (…) siempre se debe realizar una modelación previa a bajo nivel donde se evidencien las relaciones entre los objetos o variables del problema a resolver y luego pasar a una modelación matemática formal, previa a la algoritmización (…) » (E-8).

« (…) muchos estudiantes no saben analizar e interpretar un problema. (…) » (E-5).

« (…) la mayoría de los estudiantes no representa adecuadamente la situación problémica a través de estructuras matemáticas. (E-18).

Los profesores reconocen la algoritmización como parte esencial del proceso de programación que se encuentra enmarcada en la etapa de elaboración del programa, como una estructura que tiene carácter generalizador, al permitir su implementación en cualquier lenguaje. También hacen énfasis en la etapa previa de análisis e interpretación de la situación problémica. Ponen en evidencia la pertinencia de diseñar un algoritmo que refleje la solución del problema que se está intentando resolver, antes de pasar a su implementación.

2. Otros aspectos que considere relevantes para el proceso de enseñanza – aprendizaje de la Programación.

« Se debería potenciar el uso de paquetes matemáticos en la computadora. » (E-5).

« El programa matemático no es el mismo que el computacional, puesto que en el primero se da una entrada y se obtiene un resultado y en el computacional se busca una generalización tal que para cualquier dato se obtenga el mismo resultado. » (E-7).

« (…) se podría potenciar la enseñanza con el uso de algún software que mostrara el funcionamiento de la computadora. » (E-2).

« (…) sería de mucha ayuda contar con software educativos para la asignatura, que permitieran ver al estudiante realmente que ocurre en la máquina cuando se diseña un algoritmo, esto le ayudaría mucho a la hora de tener que abstraerse para diseñar un algoritmo. » (E-6).

« (…) enseñar a resolver problemas de programación no puede ser con las instrucciones solamente, sino a través de las abstracciones, para que el estudiante tenga una idea del proceso. » (E-7).

« (…) se debería en todo momento dar nociones a los estudiantes de cómo optimizar el algoritmo, aunque sean muy básicas, para crear una cultura. » (E-6).

Los profesores consideran como aspectos relevantes para el proceso de enseñanza – aprendizaje de la Programación, el uso de paquetes matemáticos, así como de otros que permitan que los estudiantes se representen el funcionamiento de la computadora cuando se diseña un algoritmo. También valoran como relevante la formación de un pensamiento abstracto en función de la programación y de habilidades para optimizar los algoritmos.

3. ¿Cómo le gustaría a usted aprender programación?

« Dedicando más tiempo a la algoritmización, por ejemplo un semestre donde solo diseñara algoritmos en pseudocódigos y diagramas de flujo (…) » (E-1).

« Adquiriendo una mejor preparación en conocimientos de Matemática para facilitar la creación de los algoritmos. (…) » (E-3).

« Logrando que en el preuniversitario se enseñara a razonar a la hora de resolver un problema. » (E-4).

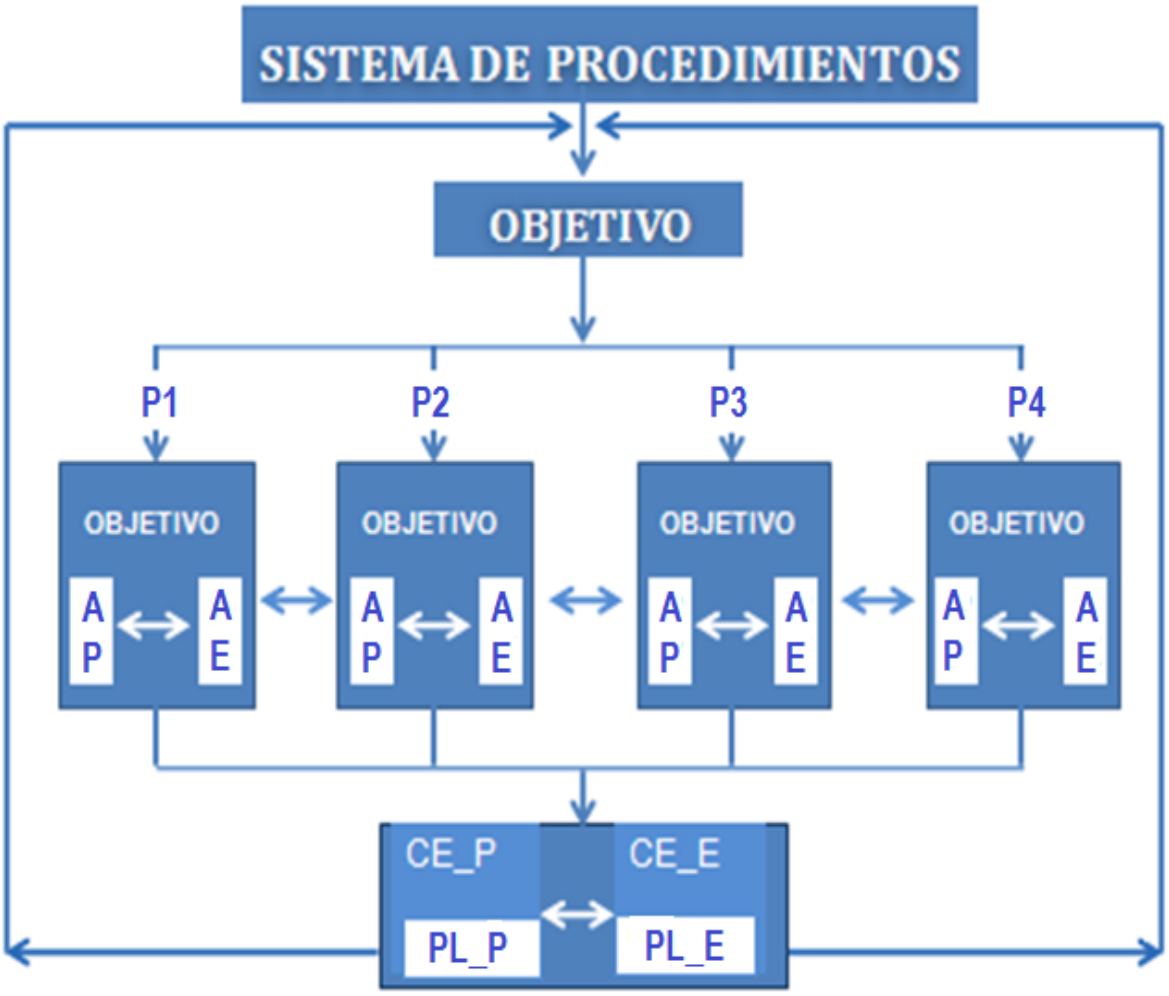
« (…) consiguiendo que en el preuniversitario se enseñara a diseñar algoritmos para la computadora, aunque no se implementasen, con el objetivo de desarrollar el pensamiento lógico (…) » (E-5).

« (…) potenciando el uso de las computadoras desde que el estudiante empieza a dar programación con más tiempo en el laboratorio. (…) » (E-20).

A esta pregunta los profesores contestaron en tres direcciones sumamente importantes, una referida al aumento del trabajo de la formación algorítmica, otra dirigida hacia el reforzamiento de la formación matemática y la última encaminada a comenzar el trabajo de programación desde enseñanzas precedentes.

ANEXO 9

ESTRUCTURA DEL SISTEMA DE PROCEDIMIENTOS DIDÁCTICOS PARA LA ALGORITMIZACIÓN COMPUTACIONAL



Leyenda			
P1, P2, P3, P4: Procedimientos del Sistema	**AP**: Acciones para el profesor	**CE_P**: Criterios evaluativos para el profesor	**PL_P**: Patrones de logro para el profesor
	AE: Acciones para el estudiante	**CE_E**: Criterios evaluativos para el estudiante	**PL_E**: Patrones de logro para el estudiante

ANEXO 10

EVIDENCIAS DE LA REALIZACIÓN DE LOS TALLERES DE SOCIALIZACIÓN

10.1 Taller con el colectivo de profesores de Programación de la carrerea de Ingeniería en Ciencias Informáticas, realizado en el Departamento Central de Técnicas de Programación de la Universidad de Ciencias Informáticas (UCI) de La Habana

UCI
Universidad de las Ciencias Informáticas

Dpto. Central de Técnicas de Programación
Docente 4, Tercer piso Telf: 837-2571, 837-2572

Fecha: 20/3/2013

Por este medio certifico:

Que el profesor **Lic. Antonio Salgado Castillo** realizó un Taller de Socialización con miembros del colectivo de programación de la Universidad de las Ciencias Informáticas (UCI). Dicho taller fue de gran importancia para el colectivo de profesores presentes ya que ayudó a caracterizar la problemática de la Enseñanza de la Programación y la Algoritmización en los primeros años de la Carrera y aportó elementos concretos para enfrentar su solución.

Para que así conste firman la presente:

Dr. Liesner Acevedo Martínez
J'Dpto. Metodológico Programación (UCI)

Dr. Natalia Martínez Sánchez
Vicerrectora de Formación (UCI).

UCI Universidad de las Ciencias Informáticas
VICERRECTORÍA DE FORMACIÓN

10.2 Taller con el colectivo de profesores de Programación de la carrera de Ingeniería en Telecomunicaciones y Electrónica de la Universidad de Oriente en Santiago de Cuba

UNIVERSIDAD DE ORIENTE
FACULTAD DE INGENIERÍA ELÉCTRICA

Santiago de Cuba, 11 de junio de 2013
"Año 55 de la Revolución"

A quien pueda interesar:

Por medio de la presente damos a conocer que el Licenciado en Ciencia de la Computación Antonio Salgado Castillo, desarrolló un **Taller de Socialización** con los profesores del colectivo de la disciplina de Computación de la carrera de Ingeniería en Telecomunicaciones y Electrónica de la Universidad de Oriente en Santiago de Cuba.

En dicho taller el aspirante expuso una síntesis de los resultados de su investigación doctoral **¨Dinámica lógico – algorítmica para la Resolución de problema de Programación computacional¨**. Los resultados de esta investigación pueden incidir positivamente aportando nuevos elementos a la forma de impartir la asignatura **Programación I** a los estudiantes de 1er año de la carrera. Además el profesor también trabajó con los estudiantes, impartiéndoles varios encuentros sobre modelación de problemas y algoritmización.

De este taller surgieron señalamientos y recomendaciones válidos para mejorar sus aportes. También se le solicitó la posibilidad de usar parte del sistema de procedimientos para la elaboración de una nueva asignatura optativa ¨Algoritmización y Matlab¨ que se comenzará a impartir en el próximo curso académico.

Sin otro asunto que tratar, firmamos la presente en Santiago de Cuba, a los 11 días del mes de junio de 2013 los abajo firmantes.

Dr. C. Ángel Enrique Cano García
Jefe del Dpto. Telecomunicaciones

Dr. C. Julio García Garay
Decano Facultad de Ingeniería Eléctrica

10.3 Taller con el colectivo de profesores de Programación de la carrera de Licenciatura en Ciencia de la Computación de la Universidad de Oriente en Santiago de Cuba

FACULTAD MATEMÁTICA COMPUTACIÓN
UNIVERSIDAD DE ORIENTE

UNIVERSIDAD DE ORIENTE

FACULTAD DE MATEMÁTICA Y COMPUTACIÓN

CONSTANCIA DE REALIZACIÓN DE TALLER DE SOCIALIZACIÓN

Santiago de Cuba, 5 de junio de 2013
Año 55 de la Revolución

A quien pueda interesar:

La presente es para hacer constar que el profesor Antonio Salgado Castillo realizó en el día de hoy un taller de socialización sobre los resultados de su tesis doctoral titulada: Dinámica lógico – algorítmica de la resolución de problemas de Programación Computacional, ante los profesores del Departamento de Ciencia de la Computación de la Universidad de Oriente y con la presencia del decano de la facultad.

Dicho taller fue de gran utilidad para el aspirante y los participantes, ya que se hicieron importantes recomendaciones que deben contribuir al perfeccionamiento de la propuesta y se decidió la introducción del aporte práctico en el primer año de la carrera, consistente en un Sistema de procedimientos didácticos para la enseñanza – aprendizaje de la algoritmización en la resolución de problemas computacionales.

Y para que así conste, firmamos la presente:

MSc. José Ramón Rojas Castro
J´ Dpto. Computación

Dr. C. Adolfo Arsenio Fernández García
Decano Facultad Matemática y Computación

FACULTAD MATEMÁTICA COMPUTACIÓN
UNIVERSIDAD DE ORIENTE

10.4 Taller con el colectivo de profesores del Departamento de Licenciatura en Ciencia de la Computación de la Universidad Central "Marta Abreu" de Las Villas

UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS
FACULTAD DE MATEMÁTICA, FÍSICA Y COMPUTACIÓN
Carretera a Camajuaní Km. 5 ½. Santa Clara. Villa Clara.54830. CUBA. Telef.: (53) (42) 281515 Fax: (53)(42) 281608

Santa Clara, 5 de marzo de 2014
"Año 56 de la Revolución"

A quien pueda interesar:

La presente es para hacer constar que el profesor MSc. Antonio Salgado Castillo realizó en el día de hoy un taller de socialización sobre los resultados de su tesis doctoral titulada: "Dinámica lógico – algorítmica de la resolución de problemas de programación computacional", ante los profesores del Departamento de Ciencia de la Computación de la Universidad Central de la Villas.

Dicho taller fue de gran utilidad para los profesores, pues se presentó un Modelo de la Dinámica lógico – algorítmica de la resolución de problemas de programación computacional y un Sistema de Procedimientos Didácticos, que consideramos se ajusta adecuadamente a las habilidades de programación que se deben formar en los estudiantes de las carreras de ciencias computacionales.

Al aspirante se le hicieron algunas recomendaciones con el objetivo de perfeccionar la propuesta en el sentido de que se introduzca algún software educativo que se sustente en los nuevos resultados investigativos y que sirva de apoyo a los profesores durante la impartición de la docencia. Se considera que la ejemplificación que se presenta, concreta toda la intencionalidad teórica – metodológica del investigador y se ajusta a los tipos de situaciones problémicas a las que se deben enfrentar los profesionales de las ciencias computacionales.

Dra. C. Gheisa Lucía Ferreira Lorenzo
Jefe del Departamento de Ciencia de la Computación
Facultad de Matemática, Física y Computación

ANEXO 11

PRUEBAS APLICADAS A LOS DOS GRUPOS BAJO ESTUDIO (CONTROL Y EXPERIMENTAL)

11.1 Preprueba aplicada a los dos grupos bajo estudio (control y experimental)

Problema: Dado un número de 5 cifras determinar si es Palíndromo.

Se debe tener en cuenta que un número Palíndromo es aquel cuyas cifras colocadas en orden inverso permiten formar el mismo número.

Ejemplo: 23432.

11.2 Postprueba aplicada a los dos grupos bajo estudio (control y experimental)

Problema: Las facultades de Ingeniería Eléctrica (FIE) y Matemática y Computación (FCMC) de la Universidad de Oriente se propusieron, en este año 2013, organizar conjuntamente la competencia de Programación de la ACM a nivel regional en el área del Caribe, en la que participaran 30 universidades. Para ello fue necesario seleccionar los mejores equipos de ambas facultades, a partir de los resultados obtenidos en las competencias previas realizadas en cada una. Debe tenerse en cuenta que la FIE posee 6 equipos y la FCMC cuenta con 4; además, cada equipo está integrado por 4 competidores y cada competidor tiene un **average** según sus resultados previos. Se conoce que cada universidad puede estar representada por 2 equipos como máximo. Sabiendo que la selección tiene en cuenta el o los equipos con mayor average, y que el average de un equipo se determina como la diferencia entre el producto de los averages de sus miembros y la suma de estos averages. Diseñe un algoritmo que permita saber:

a) ¿Cuál es el equipo con mayor average?

b) ¿Cuál es el competidor con mayor average entre las dos facultades?

c) ¿Cuál es la facultad con mejores resultados?

d) ¿Cuáles serán los equipos seleccionados?

Nota: La facultad con mejores resultados será aquella que tenga el equipo con mayor average y el competidor con mayor average en ese equipo.