Luz. Año XVII. (3), pp.30-43, julio-septiembre, 2018. Edición 76. III Época. ISSN 1814-151X
http://luz.uho.edu.cu

La algoritmización: requisito necesario para la solución de problemas con el empleo de un lenguaje de programación

The Algorithmization: Essential Requirement for Solving Problems with the Use of a Programming Language

[*]Juan Carlos Fonden-Calzadilla

[**]Mavis Lis Stuart-Cárdenas

[***]Lianne Rodríguez-Matos

[*]CUJAE (Universidad Tecnológica José Antonio Echeverría). La Habana. Cuba. Licenciado en Educación, especialidad Matemática. Doctor en Ciencias Pedagógicas. Profesor Auxiliar. fonden1980@gmail.com

[**]CUJAE (Universidad Tecnológica José Antonio Echeverría). La Habana. Cuba. Ingeniería Industrial e Ingeniería Informática. Máster en Informática Aplicada. Profesora Auxiliar. mavis@ind.cujae.edu.cu

[***]Empresa de ventas de autos. Grupo Parvi. Departamento de Tecnologías Informáticas. Recife. Brasil. Licenciada en Ciencias de la Computación. Analista de Bases de Datos. lianrmatos@gmail.com

Resumen

El presente trabajo busca argumentar la importancia del diseño y la aplicación de algoritmos en la solución de problemas docentes que se resuelven con el empleo de un lenguaje de programación. Para el logro de este objetivo se aplicaron métodos teóricos, entre ellos, la revisión y el análisis documental, la sistematización y el método histórico lógico, junto con las vivencias de los autores como profesores de programación en diferentes contextos universitarios. Se argumentó críticamente la necesidad de elaborar un algoritmo antes de escribir la primera línea de código, teniendo en cuenta la identificación de errores fatales ocurridos en diferentes partes del planeta, incluidos los errores lógicos, considerados los más difíciles de resolver al programar la solución de un problema y se explicó cómo evitarlos para que el programa se comporte de la manera esperada, concluyendo que esto se garantiza si una parte del código se encarga del tratamiento de eventos inesperados. Al mismo tiempo, se evidenció que para alcanzar el objetivo final, la creación de un buen algoritmo es tan interesante y vital como la creación de un programa con el uso de un lenguaje concreto.

Palabras clave: algoritmo; lenguaje de programación; estrategia; pseudocódigo; diagrama; solución de problemas

Abstract

The objective of this article is to argue the importance of the design and application of algorithms in the solution of teaching problems that are solved with the use of a programming language. To achieve this objective, theoretical methods were applied, including review and documentary analysis, systematization and logical historical method, together with the experiences of the authors as teachers of programming in different university contexts. The need to elaborate an algorithm before writing the first line of code was critically argued, taking into account the identification of fatal errors occurring in different parts of the planet, including logical errors, considered the most difficult to solve when programming the solution of a problem and explained how to avoid them so that the program behaves in the expected way, concluding that this is guaranteed if a part of the code deals with the treatment of unexpected events. At the same time, it was evidenced that, to reach the final goal, the creation of a good algorithm is as interesting and vital as the creation of a program with the use of a concrete language.

Key words: algorithm; programming language; strategy; pseudocode; diagram; problems solution.

Introducción

Recibido: 27 de marzo de 2018/Aceptado: 2 de abril de 2018/Publicado: 1 de julio de 2018

El aprendizaje de la programación, componente fundamental en la formación informática del Ingeniero, es un proceso continuo, complejo y planificado. Su éxito depende de que se proyecten, organicen, ejecuten y controlen sistemas de acciones con una adecuada relación teórico-práctica, donde se integren elementos motivacionales, afectivos, cognitivos y valorativos, en correspondencia con las exigencias y necesidades que plantea el plan de estudio y los objetivos que persigue la asignatura (Fonden, 2015). La programación requiere de diversas estrategias para su mejor comprensión (Stuart, 2009).

Por su importancia en la formación del pensamiento lógico, la solución de problemas y su aporte al desarrollo de la capacidad de entender conceptos y abstracciones matemáticas en los estudiantes, países como Francia, Reino Unido, Estonia, Alemania y Estados Unidos en 2014 introdujeron gradualmente la enseñanza programación en la enseñanza primaria. Otros países como Finlandia, Israel, Corea del Sur, Nueva Zelanda o Grecia llevan tiempo trabajando en programas pilotos. (Velasco, 2014) Por su complejidad, expertos debaten sus beneficios en la formación intelectual del estudiante, y desde qué edad es más conveniente introducirla a la vez que se requiere de una apropiada formación de los docentes.

Materiales y métodos

Se realizó un estudio bibliográfico exploratorio con el empleo de fuentes periodísticas de diarios digitales, opiniones de expertos en las redes sociales y páginas web, luego se procedió al análisis documental de diversas fuentes que se dedican al estudio de la programación, desde su enseñanza, hasta la práctica diaria de esta actividad.

La investigación se complementó con la experiencia vivencial de los autores como profesores de programación en diferentes contextos universitarios en los cuales se registraron datos e informaciones de exámenes y su desenvolvimiento en clases prácticas y de laboratorios. En su totalidad se consultaron más de 30 fuentes bibliográficas, en donde se adicionan tesis de doctorado, artículos científicos, videos y tutoriales.

La sistematización de los estudios realizados sobre la enseñanza de la programación y el análisis documental permitieron al autor identificar las tendencias actuales que se producen en la ocurrencia de errores y eventos inesperados en este proceso.

La modelación se empleó al elaborar el diagrama UML y el seudocódigo, ambos presentados en la introducción de este trabajo.

Para el conocimiento de las distintas etapas del desarrollo y evolución del proceso de enseñanza-aprendizaje de la Programación, sus antecedentes, contradicciones y desarrollo hasta el presente, en diferentes partes del planeta, el autor se valió del método histórico-lógico. Mediante el método de enfoque de sistema se identificaron las relaciones que existen entre los distintos elementos que conforman la Programación como actividad intelectual, su proceso de enseñanza y aprendizaje, así como las concatenaciones manifiestas entre los resultados que se obtienen a partir de aplicar la algoritmización antes de ocuparse de escribir códigos en un entorno de Programación o programar directamente prescindiendo de este paso en la solución de un problema.

Los procedimientos lógicos del pensamiento análisis-síntesis, así como la inducción-deducción, facilitaron a los autores elaborar un conjunto de recomendaciones y conclusiones, aplicables a estudiantes, profesores y programadores. La observación de clases prácticas y de laboratorios, permitió a los autores registrar los modos de actuación de los estudiantes, los aciertos y desaciertos, en la interpretación problemas, al algoritmizar, elaborar softwares y explicarlos.

Resultados y discusión

Identificación de situaciones problemáticas relacionadas con la programación

Un estudio de los resultados docentes en los cursos escolares 2010-2011 hasta el curso 2015-2016 de las facultades de Ingeniería Informática e Ingeniería Industrial del Instituto Superior Politécnico José Antonio Echeverría en la Habana y la Escuela Superior Politécnica José Eduardo Dos Santos de la provincia Bie en Angola, en los cursos escolares 2015-2016, 2016-2017 y 2017-2018, en las asignaturas Introducción a la Programación y Programación Orientada a Objetos hizo posible la identificación de las siguientes dificultades:

Deficiente interpretación de los problemas planteados; Mala aplicación de las estructuras algorítmicas necesarias para resolver cada problema; Tendencia a programar directamente en la computadora, sin antes realizar un algoritmo; Solución del problema mediante el método “tanteo y error” hasta llegar a la solución; Memorizar códigos completos o fragmentos de código en vez de emplear el razonamiento lógico para obtenerlos; Aprender a programar en un lenguaje de programación ignorando la existencia su normas generales y elementales adaptables a todo lenguaje; No tener en cuenta los posibles errores que se presentarán en la ejecución del software; Pretender aplicar la lógica de solución de un tipo de problema a otro muy diferente sin un previo análisis; Dificultades para explicar un algoritmo o programa; Poco hábito de realizar la corrida de un algoritmo introduciéndoles una diversidad de datos antes de codificarlo; En el primer examen parcial, en las citadas instituciones universitarias los

resultados oscilan entre un 47 y un 51% y en los segundos exámenes parciales los resultados crecen ligeramente entre un 54 y un 63%.

Además, se identifican también otras dificultades en otros contextos universitarios a nivel internacional que pudieran estar relacionadas con el desconocimiento de los pasos necesarios para la solución de un problema, no solo en la asignatura Programación sino en todas las áreas del conocimiento. En este sentido una publicación chilena expresa que "la deserción, cercana al 47% que se produce en los primeros años de ingeniería, sugiere necesariamente una mirada crítica y analítica de este fenómeno" (Mora-Rivera y otros, 2017; Lacoa y otros, 2016) y un estudio refleja que entre un 20 y un 80% de los estudiantes de USA abandonan sus primeras clases de programación debido a la dificultad que enfrentan para aprender a esta disciplina. Otras publicaciones se refieren a problemas análogos en México, Argentina y Brasil. Según (Revista Universia, 2017) al menos 15 universidades españolas tienen un alto grado de deserción en ingeniería informática, cuantificado en un 22,5% durante el primer año para el curso escolar 2015-2016.

Un artículo elaborado sobre investigaciones en dos universidades de Costa Rica expresa que las mujeres ingresan a las carreras universitarias de informática con menos experiencia previa en programación que los hombres, lo que aumenta la probabilidad de que abandonen la carrera, con lo cual pierden la oportunidad de obtener un empleo bien pagado y con un excelente potencial de desarrollo profesional. (Mora-Rivera, 2017)

Algoritmos. Definición. Ejemplos.

Se entiende por **algoritmo** un número finito de pasos lógicos y ordenados, dirigidos a la solución de un problema, que conllevan a la realización de acciones concretas para el procesamiento de un conjunto de datos de entrada a partir de un estado inicial y la obtención de una solución para dicho problema.

Un algoritmo es secuencia de instrucciones ordenadas para resolver un problema (Fonden, 2015; Stuart, 2009; Insuasti, 2016).

Los algoritmos, en dependencia de los objetivos para los que ellos estén diseñados se clasifican en cualitativos o cuantitativos:

Los cualitativos son aquellos que sus secuencias de pasos no realizan ningún cálculo numérico. Ejemplo: ¿Cómo bañar un bebé? ¿Cómo llamar la atención de una persona?

Los cuantitativos son aquellos en las que sus secuencias de pasos realizan cálculos numéricos. Ejemplo: ¿Cómo sumar dos números? ¿Cómo calcular el área de un triángulo?

La solución de un problema demanda la interpretación y análisis de los elementos que lo integran, luego, se diseña un modelo (secuencia de pasos lógicos) que lo solucione, el cual debe ser verificado con juegos de datos significativos que abarquen todo el posible rango de valores de entrada, en el que se efectúa un registro de su solución, en una hoja de papel o un editor de texto, para observar posibles modificaciones y fallas detectadas al introducir los datos, en el procesamiento de los mismos y la obtención de los resultados. Las acciones antes descritas se conocen como prueba del algoritmo, lo que posibilita evidenciar que se ejecutarán las tareas para las que se ha diseñado, con resultados correctos y deseados. (Fonden, 2015; Insuasti & Lacoa y otros, 2016).

Lo anteriormente expuesto excede las problemáticas computacionales y es aplicable a una variedad de problemas y situaciones que se presenten tanto en la docencia como en otras actividades del mundo real, y aún más, en escenarios de altas complejidades, en los que se precisa reflexionar e investigar, asumiendo como premisa que los estudiantes trabajen en equipos donde se complementen los conocimientos y habilidades desarrolladas y elaboren las estrategias adecuadas, (Caselli, 2010) capaces de concebir una solución gradual al problema, que permita ofrecer una respuesta definitiva, concreta.

Stuart, 2009; Fonden, 2015 y otros autores destacan tres momentos esenciales en la elaboración de un algoritmo, ellos son la *entrada de datos*, el *procesamiento* y la *salida*.

La entrada significa que el algoritmo recibe datos para luego ser procesados y que responde a la pregunta: ¿Qué datos o informaciones se necesitan?

En el procesamiento se emplean formulas, se realizan cálculos y se toman decisiones, como resultado de preguntarse: ¿Cómo podemos llegar al resultado esperado? Y finalmente la salida de la información que responde a la interrogante ¿Qué se desea obtener? Todo algoritmo debe tener una salida, o sea, genera resultados del procesamiento, los cuales guardan una relación específica con la entrada, aunque la salida no sea necesariamente un valor, pero que como resultado generan cambios en la información con la que se cuenta.

La estrategia propuesta por (Fonden, 2015) contiene los pasos esenciales para resolver un problema con el empleo de un lenguaje de programación, ellos son:

Leer, analizar el enunciado e identificar los datos disponibles para la entrada, procesamiento y salida; en segundo lugar, identificar las expresiones aritméticas, relacionales, lógicas o de otro tipo que se debe aplicar; seguidamente, elaborar el pseudocódigo y/o diagrama de actividad UML que modela la solución del problema y mostrar la información o el resultado esperado; luego, hacer una corrida con

suficientes datos seleccionados por los programadores, que permita corroborar la validez y eficiencia del algoritmo y por último, escribir el código, en correspondencia con el algoritmo realizado. Finalmente, una vez revisada su sintaxis y la lógica de solución, compilar y ejecutar el programa con diversos datos.

El seudocódigo y el diagrama de actividades

Modelar la solución de un problema mediante un diagrama de actividades, de flujo o un pseudocódigo, significa la realizar un diseño en un lenguaje natural, sin preocuparse por la sintaxis de ningún lenguaje de Programación, aunque cumpliendo con las pautas fijadas en el lenguaje algorítmico, que es el conjunto de símbolos, palabras y reglas concretas empleadas para explicar procesos o resolver un problema**,** que conlleva a identificar la secuencia de pasos ordenados, precisos y finitos para introducir datos, procesarlos y proporcionarle la salida deseada (Cherry, 2017).

El pseudocódigo está considerado como una descripción informal de alto nivel de la solución de un problema que puede ser escrita en la lengua nativa del que la empleará o en el idioma Inglés si así lo prefiere (Caselli, 2010). Se ha expresado en (Fonden, 2015) que el pseudocódigo es un lenguaje natural, informal, que auxilia al programador en el desarrollo de los algoritmos, para luego, con el empleo de un lenguaje de programación, ser convertidos en programas. Los programas en pseudocódigos no son ejecutados en la computadora, pero ayudan al programador a razonar antes de escribirlos en un lenguaje, como el C#, C++ o Java.

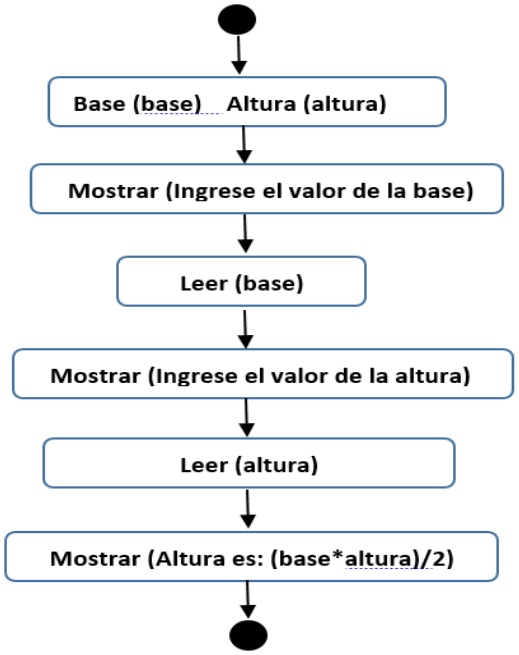
El modelar un algoritmo en seudocódigo permite al programador usar términos propios. Generalmente se emplean palabras que denotan las acciones a realizar como: Entrar, Hacer, Repetir, Mientras, Calcular, Si, Entonces, Sino, Mostrar, Inicio y Fin, y para las operaciones matemáticas y comparaciones, se usan símbolos similares a los del álgebra y la lógica matemática, ejemplo: +,-, *, /, y, <, >, =, <=, >=.

La solución de un problema, modelada mediante un diagrama de actividades UML (Vidal y otros, 2012) o un diagrama de flujo, es la representación gráfica de un algoritmo o de una porción de este. Se construye utilizando símbolos, como pequeños círculos rellenos, rectángulos redondeados, rombos y líneas que unen sus componentes, llamadas líneas de flujo.

Tanto los seudocódigos como los diagramas obedecen a ciertas reglas o principios básicos para su elaboración. Debe destacarse que se considera que la utilización de diagramas para modelar la solución de un problema tiene ventajas con relación al seudocódigo, esencialmente cuando son construidos para estudiantes de la enseñanza primaria, media y media superior. Numerosas investigaciones han

mostrado que el aprendizaje visual es uno de los mejores métodos para desarrollar habilidades del pensamiento. En la educación superior la tendencia es al empleo de seudocódigos para la modelación algorítmicamente de una solución.

En el siguiente ejemplo se calcula el área de un triángulo, conocida su base y la altura, se muestran las soluciones mediante un diagrama de actividades UML y en pseudocódigo:



Inicio
Base (base), Altura (altura)
Mostrar (“Ingrese el valor de la base”)
Leer (base)
Mostrar (“Ingrese el valor de la altura”)
Leer (altura)
Mostrar (“Área del triángulo es:”, (base*altura) /2)
Fin

Fig. 1. Algoritmo representado mediante un diagrama de actividades UML y pseudocódigo que permite conocer el área de un triángulo a partir de la base y la altura. (Elaboración propia).

El lenguaje de modelado UML permite abstraer los elementos que componen el problema en sí mismo, entre ellos, el valor de la base y de la altura; el cálculo del área y finalmente imprimir el resultado esperado. Todas las actividades ocurren de forma secuencial, una tras otra, sin que exista la toma de alguna decisión, dentro de un inicio y un fin para cumplir con la finitud de un algoritmo. Se puede inferir algo muy similar en el modelado mediante un pseudocódigo, en el que existe correspondencia entre los pasos esenciales, la entrada de datos, el procesamiento y la salida de información.

Lo antes expuesto constituye un paso intermedio entre el problema y el código que da solución al mismo. A continuación, el código realizado en el lenguaje de programación C#.

```
float valorBase, valorAltura, areaTriangulo;
Console.WriteLine("Ingrese el valor de la base");
valorBase = float.Parse(Console.ReadLine());
```

```
Console.WriteLine("Ingrese el valor de la altura");
valorAltura = float.Parse(Console.ReadLine());
areaTriangulo = (valorBase * valorAltura) / 2;
Console.WriteLine("el áera del triángulo es " +areaTriangulo);
Console.ReadKey();
```

Obsérvese que para **Mostra**r se emplea la instrucción **Console.WriteLine()**. Para **Leer** se emplea la instrucción **Console.ReadLine()**. El procesamiento es el mismo que se realiza en ambos algoritmos, **areaTriangulo = (valorBase * valorAltura)/2.** Se produce una transición natural de los algoritmos anteriores al código de programación que resuelve este problema.

Del algoritmo al lenguaje de programación. ¿Por cuál comenzar?

La programación es más que el empleo de un lenguaje determinado, confundir ambas cosas es desconocer que existe una base conceptual y metodológica común y aplicable a todos los lenguajes. No es aconsejable admitir como idea absoluta el análisis y desarrollo de soluciones algorítmicas puras, independiente a cualquier lenguaje de programación pues aquí se ignoran sus particularidades y este equívoco puede retrasar el aprendizaje. Una posición coherente será el adoptar ambas ideas, desarrollar una solución algorítmica que pueda ser fácilmente transferible a un lenguaje. Es decir, algoritmizar teniendo en cuenta los fundamentos teóricos y metodológicos del diseño de programas, con cierta independencia de los lenguajes, pero sin ignorar las características de los mismos.

El asunto en cuestión es lograr la mayor aproximación del pseudocódigo que soluciona el problema al lenguaje de programación que empleas (García, 2009).

Según una encuesta realizada en Internet en 2017 los lenguajes de programación más empleados en la actualidad son Java, C, C++, Python y C#. La selección del lenguaje de programación para la puesta a punto el algoritmo realizado depende de su complejidad y de los conocimientos del programador. Empeñarse en emplear el lenguaje que más se usa y no el que más se ajusta a tu proyecto o el que más dominas puede traer resultados indeseados (Zahumenszky, C. 2013).

Los lenguajes de programación más recomendados para comenzar y desarrollarse en esta tarea son los orientados a objetos (Universia Noticias, 2017; Zahumenszky, 2013) ya que explotan la tendencia natural que tiene el ser humano a describir, clasificar y ordenar cosas o entidades, a partir de que los objetos pueden ser agrupados según sus características, tal como se observa en la vida cotidiana. El universo está formado por objetos relacionados y agrupados entre sí, según criterios establecidos. Los

objetos tienen un comportamiento, entre ellos se realizan operaciones y se agrupan en una estructura denominada clases. La programación orientada a objetos modela el mundo real, cualquier entidad, una circunferencia un automóvil o un libro, puede ser identificada como un objeto.

Aplicaciones. Algoritmos eficientes. Errores costosos.

Los softwares a la vez que mejoran la calidad de vida de millones de seres humanos, cuando fallan pueden traer consecuencias indeseadas. Una revista publicó en 2018 los desastres más famosos relacionados con el Software, probablemente, por fallas en sus algoritmos bases, que han causado pérdidas humanas y materiales. (Nakao, 2004) Entre ellos se destacan:

Liberación anticipada de delincuentes: En octubre de 2005 se informó de que 23 presos del Departamento de Correccionales de Michigan (EE.UU.) habían sido puestos en libertad antes de que finalizara su condena debido a un fallo de programación informática.

Aerolínea American Airlines: En 2013, un error de programación provocó el caos en la compañía de aviación American Airlines. La unión de dos sistemas, como resultado de la fusión de varias compañías aéreas, originó un fallo en el sistema de reserva de pasajes.

Infraestructura: apagón en el noreste de EE.UU: En agosto del 2003 varios estados del noreste de EE.UU. y la provincia canadiense de Ontario se quedaron sin luz debido a un corte de energía resultado de un accidente local. El accidente pasó desapercibido a causa de un fallo del **software de vigilancia** del funcionamiento de General Electric Energy y provocó una cadena de errores.

Errores comunes en la programación. ¿cómo evitarlos? Recomendaciones.

Según (Rancel, 2015) se considera error a cualquier circunstancia que da lugar a un malfuncionamiento de un programa. Por malfuncionamiento se entiende una respuesta no deseada: desde meros problemas estéticos hasta graves fallos o bloqueos. Los errores fundamentales que se comenten al programar son los siguientes: Errores de sintaxis; Errores por procesos no válidos; Errores lógicos; Errores aritméticos. (Nakao, 2004; Microsoft Soporte, 2015, Fernández y otros, 2014)

Los errores de sintaxis ocurren cuando se infringen las normas de escrituras del lenguaje. Pueden ser palabras mal escritas o la ausencia de elementos como un punto y coma o una llave.

Los errores por procesos no validos observado por los autores en las clases de programación son los siguientes: asignar un valor a la variable que no coincide con el tipo declarado para ella; Emplear variables no declaradas en el programa; No declarar una biblioteca necesaria para el uso de funciones matemáticas; Introducir un dato incorrecto al correr el programa; Confundir el operador de

comparación con el operador de asignación; Invocar a funciones que no existen o enviar parámetros incorrectos.

Los errores lógicos se producen porque el algoritmo diseñado contiene fallas; o cuando la lógica misma de la programación sobre la que se basa la generalidad del programa es defectuosa. Estos tipos de errores pueden necesitar de un cambio radical en su enfoque para encontrar la solución apropiada y profundizar en el diseño algorítmico. Un programa puede correr perfectamente y mostrar resultados alejados de lo que se desea. Esto puede suceder por errores de lógica y trabajar mediante el tanteo y error, sin una reflexión previa.

Un error aritmético puede ocurrir por causa una división por cero o por no escoger la función aritmética apropiada o por implementar una fórmula incorrecta. Este tipo de error está también ligado al error lógico. Se pueden producir otros errores que tienen implicación en los cálculos matemáticos, entre ellos: Cálculos con variables no numéricas, cálculos entre variables de diferente tipo y operaciones entre variables de diferentes longitudes.

Otros errores menos graves son ventanas sobrecargadas de componentes gráficos, problemas ortográficos, la no validación de una entrada de información que produce la ruptura abrupta de la corrida de un programa.

El análisis documental, sistematización de la literatura consultada y la experiencia de los autores posibilita afirmar que para evitar los errores de programación se necesita:

De una fase intensiva de pruebas y depuración, aun cuando tu planificación y algoritmización sean muy eficientes; Diseñar y verificar módulos o algoritmos independientes antes de **realizar integraciones,** que habrá que verificar también; Integrar partes no verificadas anteriormente, que supone crear estructuras de difícil comprobación y corrección.

Otras recomendaciones para estudiantes, profesores y programadores:

Realiza preguntas, trabaja en equipos y consulta bibliografías; Valora el código escrito por otros programadores; Reutiliza lo que se ha hecho y transfórmalo para que se adapte a tus necesidades; E**scribe comentarios en el código cuando sea necesario, sin exagerar, ya que puede servirte de ayuda y** es una práctica esencial en ambientes colaborativos, para que otros capten lo que se hace o para explicar el porqué de cierta lógica; Que el código sea legible, entendible. Las variables y funciones deben tener nombres explicativos o descriptivos, relacionados con el tema en cuestión, aunque sean extensos y mantener un estándar al nombrar las variables, constantes, funciones, métodos, clases y propiedades; Al diseñar un algoritmo debe tenerse en cuenta todas las situaciones posibles que

puedan presentarse durante su ejecución; Una forma acertada de evitar la ocurrencia de errores lógicos es dividir algoritmos grandes y complejos en módulos o funciones pequeñas, y así cualquier error de tipo lógico pueda identificarse fácilmente para su posterior corrección (Rancel, 2015; Suárez et al. 2015)

La introducción de la disciplina **Diseño de Algoritmos** en los cursos escolares 2016, 2017 y 2018 en la Escuela Superior Politécnica José Eduardo Dos Santos de Bie en Angola produjo un salto cuantitativo y cualitativo en las primeras evaluaciones parciales con relación a los cursos anteriores en donde se comenzaba la enseñanza de la programación directamente con el Paradigma Orientado a Objetos, aun cuando los estudiantes desconocían las estructuras fundamentales para programar: secuencial, condicional y repetitiva y además, sin conocimientos previos de variables, datos de entrada, procesamiento y salida de la información. Los resultados parciales en cursos anteriores eran inferiores al 45% de promoción y el estado anímico y motivacional de la mayor parte de los estudiantes era considerado de muy negativo.

En 2016 se obtuvo un 57% de aprobados, en 2017 se obtuvo un 65% y en el actual curso escolar se obtuvo en el primer control parcial un 64%. Datos que expresan la mejoría que se produce al desarrollar primeramente el pensamiento algorítmico de los estudiantes antes de ir a la programación en un lenguaje determinado.

Conclusiones

Para resolver un problema con el empleo un lenguaje de programación es necesario realizar rigurosamente, como mínimo, los siguientes pasos: Definición del problema; análisis y diseño del algoritmo que los soluciona, expresado en un pseudocódigo, diagrama UML o de flujo; Todos los algoritmos, en su estructura principal, constan de tres etapas o pasos: entrada; proceso o procesamiento y salida de la información; En el proceso de programar la solución de un problema se comenten errores lógicos, aritméticos, de sintaxis, por procesos no válidos, y otros menos graves. Los errores lógicos pueden evitarse si se realiza una planificación cuidadosa durante las fases de algoritmización y codificación. Es labor de los programadores predecir la existencia de errores y evitarlos; La creación de un buen algoritmo es tan interesante y necesaria como la creación de un programa con el empleo de un lenguaje concreto; En el proceso de programar la solución de un problema influyen disímiles factores como la capacidad intrínseca del programador, sus conocimientos sobre el lenguaje, entrenamiento, hábitos o costumbres, sus ideas o creencias, inclinaciones personales, su estado emocional, actitud, autoconfianza y autoestima.

Referencias bibliográficas

Cristian, L., Cabeza, C., Parra, J., Lopez, L. (2015). Experiencias prácticas con el uso del lenguaje de programación Scratch para desarrollar el pensamiento algorítmico de estudiantes en Chile. *Formación universitaria, 8,* (4). DOI: http://dx.doi.org/10.4067/S0718-50062015000400004

Caselli Gismandi, H. (2010). *Manual de Algoritmos y Estructuras de Datos.* Universidad Nacional del Santa. Facultad de Ingeniería. Escuela Académico Profesional de Ingeniería de Sistemas e Informática. Chimbote. México. Recuperado de http://biblioteca.uns.edu.pe/saladocentes/archivoz/publicacionez/manual_estructura_de_datos_2010___h._caselli_g.pdf

Fernández, C., Pérez, J. R., Paule, M, & Álvarez, V. (julio de 2014). Aprendizaje de la programación guiado por los errores de compilación. *Actas de las XX JENUI,* pp.371-378. Recuperado de: https://upcommons.upc.edu/bitstream/handle/2099/15498/P371fe_apre.pdf?sequence=1&isAllowed=y

Fonden, J.C. (2015). Seudocódigos, diagramas UML y códigos en el lenguaje de programación C# para la solución de problemas. *Revista Ingeniería Mecánica*. pp. 1-10. Recuperado de: www.ingenieriamecanica.cujae.edu.cu/index.php/revistaim/pages/view/monografia

Insuasti, J. (2016). Problemas de ensino e aprendizagem dos fundamentos de programação. *Educación y Desarrollo Social, 10,* (2), 234-246.

Cherry, K. (june 7, 2018). Problem Solving in Pyschology. *Verywellmind.* Retrieved from: https://www.verywellmind.com/what-is-an-algorithm-2794807

García López, J. C. (1 de diciembre de 2008). Algoritmos y programación. Guía para docentes. *Eduteka.* Recuperado de: http://www.eduteka.org/GuiaAlgoritmos.php

Hossian, A., Cejas, L., Carabajal, R., Echeverría, C., Olivera, V & Alveal, M.. (2015). Aplicación de Tecnologías Inteligentes para el Estudio de Conductas de Robots Móviles en Ambientes de Trabajo con Obstáculos Fijos. *Revista Latinoamericana de Ingeniería de Software, 3,* (5), 197-205. Recuperado de: http://revistas.unla.edu.ar/software/article/view/806/842. ISSN 2314-2642.

Lacoa, R. F., Lacoa, J. & Blai, A. (12 de octubre de 2016). La enseñanza de Lenguajes de Programación en la Escuela: ¿Por qué hay que prestarle atención? *Fundación Telefónica.* Recuperado de: http://www.fundaciontelefonica.cl/2016/10/12/la-ensenanza-de-lenguajes-de-programacion-en-la-escuela-por-que-hay-que-prestarle-atencion/

Microsoft Soporte. (2015). *INFO: Errores comunes de programación en lenguaje C. Traducción automática en páginas de Microsoft*. Recuperado de: https://support.microsoft.com/es-es/kb/22321/es

Mora-Rivera S., Coto-Chotto M. & Villalobos-Murillo J. (enero-abril de 2017). Participación de las mujeres en la carrera de Ingeniería Informática de la Universidad Nacional y su desempeño en los cursos de programación. *Educare Electronic Journal, 21,* (1), 1-22. Recuperado de: http://oaji.net/articles/2017/2279-1490203031.pdf, doi: http://dx.doi.org/10.15359/ree.21-1.12

Nakao, Y. (25 de agosto de 2014). 10 errores informáticos que provocaron catástrofes. *RT*. Recuperado de: https://actualidad.rt.com/actualidad/view/138158-catastrofes-programacion-culpa-software-computadora

Rancel, M. R. (2015). Diseño de algoritmos en programación: del pseudocódigo al programa. Resolución de problemas (CU00224A) Entrega nº23 del curso Bases de la programación Nivel II. *APR.* Recuperado de: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=247:diseno-de-algoritmos-en-programacion-del-pseudocodigo-al-programa-resolucion-de-problemas-cu00224a&catid=36&Itemid=60

Universia. (27 de junio de 2017). Ingeniería es el área con mayor tasa de abandono escolar en España. Ciencia y tecnologías. Recuperado de: http://noticias.universia.es/ciencia-tecnologia/noticia/2017/06/27/1153624/ingenieria-area-mayor-tasa-abandono-escolar-espana.html

Suárez Palma., A. C. & Sarmiento Porras, R. E. (abril-junio de 2015). Estado del arte sobre experiencias de enseñanza de programación a niños y jóvenes para el mejoramiento de las competencias matemáticas en primaria. *Revista Mexicana de Investigación Educativa, 20,* (65), 607-641. Recuperado de: http://www.redalyc.org/articulo.oa? id=14035408013

Stuart, M.et al. (2009). *Elementos básicos para la elaboración de algoritmos. Folleto docente. 2009.* Facultad de Ingeniería Industrial (CUJAE), La Habana. [Disponible en FTP de la Facultad de Ingeniería Industrial]

Velasco, J. J. (19 de agosto del 2014). Niños programadores: para qué sirve la enseñanza de programación en las escuelas. *Eldiario.es.* Recuperado de: https://www.eldiario.es/turing/Ninos-programadores-ensenanza-programacion-escuelas_0_293970921.html

Vidal Silva, C., Schmal, R., Rivero, S., & Villaroel, R. (2012). Extensión del Diagrama de Secuencias UML (Lenguahe de Modelado Unificado) para el Modelado Orientado a Aspectos. *Información Tecnológica, 23,* (6), 51-62 Recuperado de https://www.researchgate.net/publication/262472600_Extension_del_Diagrama_de_Secuencias_UML_para_el_Modelado_orientado_a_Aspectos