ISSN 1390 – 9770

Rev. SINAPSIS, Edición Nº 13, Vol. 2, Diciembre 2018

Visualización de software, una alternativa efectiva en la enseñanza de la programación de computadoras

Visualization software, an effective alternative in teaching computer programming

Visualización de software

Karina Virginia Mero Suarez. Ingeniera [(1)]

Edwin Joao Merchán Carreño. Ingeniero [(2)]

Carlos Renán Mero Suárez. Ingeniero [(3)]

[(1)] Carrera Ingeniería en Sistemas Computacionales, Universidad Estatal del Sur de Manabí, Manabí, Ecuador, karinaunesum@yahoo.com

[(2)] Carrera Ingeniería en Sistemas Computacionales, Universidad Estatal del Sur de Manabí, Manabí, Ecuador, joaounesum@yahoo.es

[(3)] Facultad de Ciencias Económicas, Universidad Estatal del Sur de Manabí, Manabí, Ecuador, carlos_mero_1980@hotmail.com

Contacto: karinaunesum@yahoo.com

Receptado 18/09/2018 Aceptado: 06/12/2018

Resumen

Las Técnicas de Visualización de Programas realizan representaciones gráficas y abstracciones de alto nivel que describen el código y los datos del programa, transformando la información tradicional en una más significativa que facilita la comprensión por el programador, lo que constituye un problema a resolver en la actualidad. El objetivo de este trabajo es analizar el papel de la enseñanza mediada por tecnología como alternativa frente al modelo tradicional de enseñanza, el impacto de las herramientas de visualización de software, así como las bases teóricas adecuadas para el uso de estos sistemas de ayuda al proceso de enseñanza-aprendizaje, que contribuyen a elevar la excelencia del proceso docente en la carrera Ingeniería en Sistemas Computacionales de la Universidad Estatal del Sur de Manabí en Ecuador. Se utilizan herramientas y estrategias para mostrar las etapas en el diseño e implementación de algoritmos, que permiten concluir que el enfoque constructivista, la enseñanza significativa y el enfoque psicogenético de Piaget, deben ser fundamentos a considerar para diseñar ambientes mediados por tecnologías que permitirán

desarrollar aplicaciones educativas eficaces en la enseñanza de la Programación de Computadoras, facilitando así la colaboración e intercambio entre los sujetos del proceso; así mismo se determina que el enfoque de Piaget es muy útil para diagnosticar lo que el alumno ya sabe y cómo ir, paulatinamente, usando sus potencialidades, incorporando los nuevos conceptos siempre en un mayor nivel de complejidad, respetando su ritmo para que pueda asimilar, acomodar y aplicar lo aprendido, desarrollando nuevas estructuras mentales.

***Palabras clave*:** programación de computadoras, visualización de programas, paradigmas pedagógicos, enfoques constructivistas.

Abstract

The Program Visualization Techniques perform graphic representations and high level abstractions that describe the code and data of the program, transforming the traditional information into a more meaningful one that facilitates the understanding by the programmer, which constitutes a problem to be solved at present. The objective of this work is to analyze the role of technology-mediated teaching as an alternative to the traditional teaching model, the impact of software visualization tools, as well as the adequate theoretical bases for the use of these systems to help the process of teaching-learning, that contribute to elevate the excellence of the teaching process in the career in Computer Systems Engineering of the Southern State University of Manabí in Ecuador. Tools and strategies are used to show the stages in the design and implementation of algorithms, which allow us to conclude that Piaget's constructivist approach, meaningful teaching and psychogenetic approach must be foundations to be considered in order to design environments mediated by technologies that will allow the development of applications effective educational in the teaching of Computer Programming, thus facilitating collaboration and exchange between the subjects of the process; Likewise, it is determined that Piaget's approach is very useful to diagnose what the student already knows and how to go, gradually, using their potentialities, incorporating the new concepts always in a higher level of complexity, respecting their rhythm so that they can assimilate, accommodate and apply what has been learned, developing new mental structures.

***Keywords*:** computer programming, visualization programs, pedagogical paradigms,constructivist. approaches

Introducción

En la época contemporánea, también llamada era de la información, el uso y desarrollo de aplicaciones informáticas se ha convertido en una necesidad vital. Sin embargo, se presentan

dificultades en el proceso de enseñanza-aprendizaje de las disciplinas relacionadas con la Programación de Algoritmos en las carreras de Sistema e Ingeniería Informática. Históricamente los estudiantes han presentado problemas para asimilar nociones matemáticas abstractas, fundamentalmente cuando ésta s incluyen la dinámica de cómo los algoritmos manipulan los datos (ACM & IEEE-CS, 2014). Reconociendo la afirmación anterior, muchos centros de enseñanza han dedicado grandes esfuerzos para resolver esta situación, implementado ambientes de aprendizaje apoyados en el uso de las Tecnologías de la Información y las Comunicaciones; así se abren las puertas al desarrollo de las técnicas de Inteligencia Artificial, el Aprendizaje Reforzado (Reinforcement Learning), el uso de Técnicas de Visualización de Programas y de las modalidades de educación virtual (e-learning), semipresencial (blended-learning) o enseñanza a través de los dispositivos móviles (mobil-learning) (Cañas, Granados, Pérez, Pérez, & Hill, 2015). Sin embargo, el reto de enseñar a programar de manera eficiente mantiene su actualidad y constituye un área de investigación importante.

Es necesario considerar, además, que hay varios factores que intervienen en el proceso de enseñanza-aprendizaje de la programación de computadoras, unos son simplemente inherentes al sujeto en sí mismo, mientras los otros tienen más relación con deficiencias en los métodos y recursos usados para enseñar (Hung & Rodger, 2016).

Teniendo en cuenta la situación anterior, el artículo propone analizar el papel de la enseñanza mediada por tecnología como alternativa válida frente al modelo tradicional de enseñanza de la programación de computadoras, el impacto de las herramientas de visualización de software, así como las bases teóricas adecuadas para el uso de estos sistemas de ayuda al proceso de enseñanza-aprendizaje.

Materiales y métodos

La investigación utilizó como métodos teóricosel histórico lógico, deductivo, análisis-síntesis y descriptivo para determinar el impacto que las herramientas estudiadas tenían en el proceso de enseñanza-aprendizaje de la programación de computadoras, así como para definir el paradigma pedagógico que apoyaba la aplicación de los ambientes mediados por tecnología en este contexto.

Desarrollo

Desde los albores de la historia humana, aprender ha sido una característica propia de cada ser humano, que ha contribuido a fundamentar las bases de su desarrollo, ya que cuando se aprende, se adquiere el conocimiento por medio del estudio, el ejercicio o la experiencia. Las exigencias de la vida moderna, dinámica, competitiva y llena de información, han dirigido a la sociedad hacia la

búsqueda de nuevos modelos, técnicas y sistemas que permitan adquirir esos conocimientos de una manera eficaz y eficiente (Soler, 2009).

En la actualidad, el proceso de enseñanza-aprendizaje se caracteriza por enfrentarse a un alumnado heterogéneo y al hecho de que existe una estrecha relación y complementación entre el desarrollo vertiginoso de las Tecnologías de la Información y las Comunicaciones (TIC) y la enseñanza, de ahí la necesidad de buscar nuevos modelos que ayuden a proyectar este proceso bajo una nueva perspectiva (Merril, 2016).

El desarrollo de medios tecnológicos está logrando no sólo cambiar los sistemas de relación ser humano – medio, sino también, instalarse como componente cultural, por lo que consideramos su utilización en la enseñanza es imprescindible. Entre ellos se encuentran las herramientas visuales de aprendizaje, que según Brian Moor and Fadi Deek (2014), pueden ser clasificadas para su análisis en dos categorías, que se corresponden con las actividades de comprensión y construcción de programas antes mencionadas por Señas and Moroni (2011):

- Las que emplean la visualización para mostrar y construir animaciones de programas que ya han sido escritos, apoyan la comprensión de programas, de conceptos abstractos y el proceso de eliminación de errores. Entre ellas se encuentran sistemas como EROSI y AnimPascal.
- Las que usan la visualización para ayudar a los estudiantes en el proceso de desarrollo de un programa, apoyando la comprensión y modelación de conceptos y procesos de un gran nivel de abstracción, análisis de programas y solución de problemas. Entre ellas se encuentran BlueJ, FLINT, BOOST y SOLVEIT.

Resultados del desarrollo de sistemas de visualización de software

Los empleos de visualización antes expuestos son mutuamente exclusivos y tienen fines completamente diferentes. A continuación, se describen algunas aplicaciones que usan estas herramientas.

EROSI: presentador Explícito de Invocaciones de Subprograma

Es un instrumento fácil de usar que intenta ayudar a principiantes a través de la visualización de programas, en particular los recursivos. La recursividad es un concepto abstracto muy difícil de entender y, por consiguiente, tiende a ser evitado o sustituido por algoritmos iterativos. George (2011), plantea que EROSI ayuda a que los principiantes obtengan un modelo mental para facilitar la comprensión y el empleo de la recursividad como una técnica de solución de problemas. Good

and Brna (2014), señalan, sin embargo, que el estudio de la recursividad tiene dos aspectos: el declarativo, relacionado con aprender qué es y el procedural que permite determinar cómo usarlo.

Se coincide con estos autores cuando indican que las dificultades de la recursividad no son eliminadas una vez que el concepto se comprende, ya que los estudiantes, aun cuando comprendan el concepto, encontrarán problemas al aplicarlo en un programa. De ahí la importancia que en esta investigación se le confiere a la integración de herramientas que apoyen la comprensión y la aplicación de conceptos abstractos.

AnimPascal: Pascal Animado.

Una de las dificultades fundamentales en el estudio de la programación, es la capacidad de visualizar y crear un modelo mental de las acciones implicadas en la ejecución de un programa. Los ambientes de programación carecen de la capacidad de animar con eficacia las acciones que el programa realiza. Para mejorar este aspecto, Satratzemi, Dagdilelis, and Evageledis (2014), introducen un ambiente de educación visual diseñado para enseñar a los programadores a desarrollar, verificar, eliminar errores y ejecutar programas.

Basado en el lenguaje Pascal, las dos funciones primarias de AnimPascal son las de animador de programas y como un mecanismo que puede ser usado para analizar varios caminos de resolución de los problemas registrados. Tiene una interfaz gráfica que es simple y fácil de usar y permite mostrar el proceso de ejecución paso a paso. AnimPascal intenta emplear la visualización y animación para ayudar a los nuevos programadores a crear un modelo mental de la ejecución de programas en una computadora. Se considera que su animación, sin embargo, es principalmente textual y se basa en el movimiento, para lo cual emplea variaciones de iluminación sobre el código y no permite visualizar objetos.

BlueJ: Ambiente Interactivo de Java.

Proporciona ejemplos, animación básica de la ejecución del código fuente, y la ejecución y prueba de clases individualmente. Tiene una capacidad (aunque limitada) para ayudar a principiantes a comprender los programas orientados a objetos que ya han sido escritos dentro de su marco. La ventaja primaria está en la visualización de los objetos de las clases y las asociaciones entre ellos. Este sistema centra la visualización en la modelación del objeto más que en la implementación del código, lo que anima a los estudiantes a explorar y experimentar convirtiéndolos en entes activos en el proceso de aprender a programar computadoras (Barnes & Kolling, 2015).

FLINT: Intérprete de Organigrama

Ziegler and Crews (2004), indican que el sistema Intérprete de Organigrama (FLINT) es un ambiente de programación instruccional, diseñado expresamente para principiantes que procuran incorporar el diseño, el desarrollo de algoritmos, las pruebas y la eliminación de errores en una herramienta integral, intenta abstraer al estudiante de la sintaxis de un lenguaje de programación específico. Sus creadores, plantean que el empleo de su ambiente ayuda al usuario a desarrollar una visión de la programación en la cual el diseño y las pruebas son partes integrales del desarrollo de un programa.FLINT se enfoca en dos tipos de programación: la icónica y la pseudo programación.

BOOST: Herramienta de Apoyo Orientado Objeto Básico

Genera un programa en Smalltalk, las principales actividades definidas son: la creación de una lista de las clases que pueden, o no, estar en el diseño del programador, a estas clases se les llama clases candidatas; permite una aproximación a la implementación. Proporciona, además, una funcionalidad muy básica de diseño no estructurada que es más útil a un programador más avanzado (Deek, McHugh, & Hiltz, 2013).Se puede considerar que, en general, no presta suficiente ayuda y orientación para ser usado con eficacia en el proceso de aprendizaje de un programador principiante.

SOLVEIT: Lenguaje Orientado a Especificaciones en un Ambiente Visual para Traducción de Instrucciones

Deek (1997) con Deek and McHugh (2013), proponen el lenguaje SOLVEIT como un ambiente integrado, diseñado para que los usuarios noveles aprendan a resolver problemas de programación usando un software diseñado para este fin. Está basado en el Modelo Común Dual desarrollado por Deek (1997), que representa la integración de un modelo cognoscitivo para la solución de problemas con las tareas necesarias para el desarrollo de programas, método usado en el diseño del Ambiente Integrado de Visualización de Estructuras de Datos, desarrollado en esta investigación.

SOLVEIT constituye, por su diseño, un sistema que estimula el aprendizaje, no dice directamente al usuario cómo solucionar el problema, sino que éste, implícitamente, aprende a solucionar

problemas y adquiere habilidades en el desarrollo de programas trabajando con el sistema y siguiendo las etapas que propone.

Varios autores como Cooper, Dann, and Pausch (2012), proponen nuevas herramientas visuales que tienen su fundamento en UML, las cuales intentan incorporar algunos de los beneficios de UML que, además, tienen la ventaja de brindar las dos facilidades expuestas en la clasificación dada por Brian Moor and Fadi Deek (2014), en lo que aventaja a las herramientas anteriormente analizadas.

Luego del análisis realizado de las diferentes técnicas y herramientas empleadas para apoyar el proceso de enseñanza-aprendizaje de la programación de computadoras, se reconoce la necesidad de integrar recursos que ayuden a la comprensión, modelación de conceptos y procesos de un gran nivel de abstracción, el análisis de programas, incorporando elementos de eficiencia y solución de problemas.

Mapas conceptuales

Una herramienta necesaria a tener en cuenta son los Mapas Conceptuales (MC) en la visualización de programas. Basándose en el aprendizaje como procesamiento de información, Joseph Novak and Gowin (1988), los introducen como una respuesta a la línea de Ausubel, Novak, and Hainesian (2000), del aprendizaje significativo dentro del marco de un programa denominado "Aprender a Aprender". En ellos, el conocimiento está organizado y representado en todos los niveles de abstracción, situando los más generales e inclusivos en la parte superior y los más específicos y menos inclusivos en la parte inferior.

En este trabajo se coincide con J. Novak (1991), al plantear que la creación de conocimiento requiere un alto nivel de aprendizaje significativo, los mapas conceptuales facilitan este proceso, por lo que resultan importantes en el aprendizaje, principalmente debido a que facilitan una rápida visualización de los contenidos de aprendizaje, favorecen el recuerdo y el aprendizaje de manera jerárquica organizada, permiten una rápida detección de los conceptos claves de un tema y sus relaciones, además de favorecer el desarrollo del pensamiento lógico.

Los materiales elaborados utilizando MC facilitan el estudio independiente, permiten que el alumno pueda explorar su conocimiento previo acerca de un nuevo tema, a la vez que integran la nueva información que ha aprendido, organizan los conocimientos a partir de las principales relaciones entre los conceptos y favorecen el trabajo colaborativo. Por su parte, Ontoria (2014), considera que

constituyen un recurso esquemático para representar un conjunto de significados conceptuales incluidos en una estructura de proposiciones. Estas pueden ser explícitas o implícitas.

La enseñanza de la programación, como parte fundamental de la formación profesional de las carreras de Ciencias de la Computación, ha sufrido cambios importantes que han estado aparejados al desarrollo de la tecnología y la enseñanza, muchos de ellos se han basado en el surgimiento de nuevos paradigmas de programación (Castillo & Barberán, 2016). Entre los principales obstáculos que aparecen para el aprendizaje y aplicación de un lenguaje de diseño de algoritmos pueden puntualizarse los siguientes:

- El alumno se ve necesitado de manejar un gran número de nuevos conceptos e integrarlos de manera significativa. En los algoritmos, las acciones complejas suelen definirse en términos de otras acciones más sencillas. Esto hace que la comprensión acabada de las acciones más simples redunde en beneficios para entender aquellas más complejas.

- Las acciones tienen dos aspectos que están estrechamente relacionados entre sí: una *sintaxis* (reglas de redacción de las acciones en un algoritmo), y una *semántica* (significado formal y preciso de una acción dada).

- El alumno se enfrenta a la necesidad de manejar un *lenguaje objeto* (para elaborar algoritmos) y un *metalenguaje* (para hablar acerca de cómo se comporta el lenguaje algorítmico).

- Existen conceptos relativamente complejos interrelacionados entre sí (Chestlevar, 2015).

En el desarrollo del proceso de enseñanza-aprendizaje de la programación de computadoras en docentes de la carrera Ingeniería en Sistemas Computacionales, Universidad Estatal del Sur de Manabí, Manabí, Ecuador, se ha comprobado que escribir un programa utilizando un Lenguaje de Programación requiere del alumno varias competencias y habilidades, que involucran básicamente la capacidad de manipular un conjunto de abstracciones interrelacionadas para la resolución de problemas. En tal sentido, el proceso de enseñanza-aprendizaje de un Lenguaje de Programación es extremadamente complejo.

Ese acercamiento prescinde, muchas veces, de una clara identificación de cómo se interrelacionan distintos conceptos teóricos entre sí, el resultado es que muchos se presentan independientemente y sólo a través de la práctica el alumno llega a interrelacionarlos. Esto puede motivar la exploración

de distintas técnicas didácticas que facilitan a los alumnos una mayor comprensión y vinculación de los temas presentados.

En programación se da el caso particular de que todo concepto expresado a través de la *sintaxis* de un lenguaje tiene su correlación con un significado operacional (semántica), y dicho significado estará definido de manera composicional, en término del significado de otros conceptos más elementales.

La incidencia de los mapas conceptuales en la pedagogía moderna para definir y organizar planes de estudio, currículos, programas de asignaturas y para la acción directa en el proceso de aprendizaje ha trascendido las aspiraciones iniciales de su creador (Cañas & Novak, 2004).

Resultados de la visualización en la enseñanza de la programación

La visualización de software y especialmente la visualización de programas, contribuye favorable y efectivamente en la comprensión de los mismos. Ayuda tanto en la tarea de interpretación del programa subyacente como en la de diseño, depuración, prueba y mantenimiento del software. Para ello se debe disponer de sistemas que faciliten la interacción hombre – máquina, en cuyo diseño se debe poner especial interés en la percepción humana, (Almeida, Blanco, & Moreno, 2013).

La visualización de software presenta un medio audio – visual interactivo multimedial cuyas bondades en el campo de la psicopedagogía han sido experimentadas. AlgunosautorescomoGuttag and Liskov (1986), Bladek and Deek (2010), Muthukumarasamy and Stasko (2012), Hennessy (2013) y Morris (2003), consideran que los sistemas no sólo deben estar diseñados por especialistas en computación sino por psicopedagogos que permitan establecer un equilibrio entre el exhibicionismo espectacular que presentan algunas visualizaciones, que a veces no pasa más que por la exhibición de los potentes recursos informáticos, y lo que el ser humano puede percibir efectivamente de ellas. Las Animaciones de Software presentan importantes beneficios educativos ya que estimulan y ayudan a los estudiantes en las distintas situaciones del aprendizaje:

- Logran un incremento de la motivación. Esto es, a través de una presentación atrayente, en la que un algoritmo desafiante se transforma en uno más accesible y menos intimidante, los estudiantes se sienten más motivados a interactuar con el material y a estudiar temas complejos.
- Facilitan el desarrollo de destrezas por la oportunidad de realizar prácticas adicionales. Los estudiantes tienen una nueva forma para experimentar los algoritmos. Además de resolver los

ejercicios en papel y escribir los programas, ellos pueden percibir visualmente los algoritmos y estudiar sus características observando e interactuando con la animación (Stasko & Lawrence, 2007).

- Asisten en el desarrollo de habilidades analíticas y promocionan las predicciones ya que los estudiantes deben coleccionar sus propios datos para el análisis del algoritmo y los subsecuentes diseños de algoritmos mejorados. Las animaciones de software ofrecen distintas ventajas comparadas con la ayuda ofrecida por la enseñanza tradicional, tal como el libro de texto y la pizarra.
- Ofrecen un buen soporte al docente y son de gran ayuda en la clase durante la explicación de la conducta dinámica de un algoritmo.
- Permiten la exploración de las peculiaridades de un software, mejorando la comprensión individual de los estudiantes cuando interactúan con el software y sus entradas, formulan hipótesis de la conducta del algoritmo para luego estudiar las acciones resultantes, verificando o refutando sus ideas. Esto los capacita para formar un modelo conceptual del algoritmo además de aprender el código. La interactividad agrega un nuevo nivel de efectividad al ambiente de aprendizaje, y es una herramienta apropiada en concepto de enseñanza ya que ella fuerza a los aprendices a tomar parte de la lección, y no simplemente a observar un movimiento. La interactividad ayuda a los estudiantes a adquirir una experiencia invalorable en la resolución de problemas (Merril, 2016; Stasko, Domingue, Brown, & Price, 2012).

Es importante el valor educativo de las técnicas de visualización de software. La animación de algoritmos y la visualización de programas ayudan a los estudiantes a comprender los conceptos de software y también a los docentes en su tarea de enseñar dichos conceptos. Distintas experiencias con respecto a la aplicación de la Visualización de Software en la enseñanza de la programación se han realizado en diversas universidades del mundo. Actualmente se emplean como recurso didáctico tanto en los cursos elementales como en los avanzados.

Aunque todas las animaciones de algoritmos tienen el mismo propósito, el uso de las mismas puede variar considerablemente. Algunas de las aplicaciones que parecen ser las más comunes, según Stasko et al. (2012) sirven para acompañar una lectura y ayudar a comprender los conceptos claves que explica el profesor durante la clase, en un laboratorio formal donde los estudiantes interactúan con las computadoras, para uso informal por los estudiantes fuera de la clase, en su tiempo libre, para ayudar a comprender más acerca de un algoritmo, para realizar un aprendizaje personalizado e individualizado, desarrollando el estudio independiente, el autoaprendizaje.

Los medios visuales, como añadido a medios verbales o textuales, presentan la información de una manera más eficiente a través de imágenes del medio verbal que ellos generan. Barbe and Milone (2015), observan que, por consiguiente, la percepción de información vía diagramas es sumamente eficaz para aprender a programar.

Discusión

La discusión sobre las herramientas y estrategias propuestas en este trabajo se basará en la comparación con autores que presentan experiencias en el uso de la Enseñanza Asistida por Computadoras para la enseñanza de la programación, de esta manera podremos corroborar la importancia de las propuestas en este trabajo.

Algunos autores privilegian el enfoque tecnológico en el uso de herramientas para enseñar a programar como expone Roman and Cox (1993), mientras que otros reconocen la necesidad de tener en cuenta bases pedagógicas y psicológicas que les han permitido identificar y clasificar gran cantidad de problemas que los principiantes afrontan al aprender a programar (Bladek & Deek, 2010).

Coincidiendo con este punto de vista, consideramos que las dificultades en el proceso de enseñanza-aprendizaje de la programación de computadoras tienen una de sus raíces en el uso de una pedagogía ineficaz, factor vital que influirá en la actuación del programador en experiencias subsecuentes. De acuerdo con Liffick and Aiken (2011), observamos que al principio los programadores pueden tener dificultades al entender un nuevo concepto, no porque sea difícil, sino debido a que depende de un concepto anterior. Otro aspecto puede ser atribuido a la inadecuada selección del problema a solucionar para alcanzar determinada competencia. Suchan and Smith (2012), plantean por consiguiente que los nuevos programadores comienzan a escribir programas y a generar el código sin realizar un proceso de análisis planificado y organizado. Pane and Myers (2013), así como, Bennedsen and Caspersen (2014), observan que las estrategias de solución al problema general deberían ser explícitamente adquiridas a través de habilidades en el desarrollo de programas.

Desde el punto de vista psicológico, la manera en que la información está representada y se manipula en una computadora provoca un desafío a la comprensión, además, algunos de los conceptos de programación, tales como la recursividad, son abstractos por naturaleza, y otros no tienen una contrapartida en el mundo real que facilite la analogía (Guzdial & Elliott, 2015).

Otra causa que influye en el proceso de enseñanza-aprendizaje de la Programación de Computadoras es la preparación en Computación y Matemáticas de los estudiantes que matriculan

en la carrera Ingeniería en Sistemas Computacionales, al menos en la UniversidadEstatal del Sur de Manabí en Ecuador. Un porciento elevado no tiene nociones formales de representación de pasos lógicos para resolver un problema (diseño de un algoritmo); hasta la enseñanza precedente han sido usuarios de aplicaciones informáticas sencillas, pero no se han preparado para enfrentar un proceso de creación de software y sus dificultades intrínsecas. A esto se une que la infraestructura computacional de las escuelas que anteceden a la universidad no les ha permitido desarrollar todas sus potencialidades en el uso de esta herramienta, que ha pasado a ser el objeto de estudio de su carrera.

A juicio de varios autores con los que concordamos, una estrategia pedagógica a emplear para la Enseñanza Asistida por Computadoras aplicada a las Ciencias de la Computación es la enseñanza significativa y el enfoque constructivista, (Chestlevar, 2015; Lezcano, 1998). Los primeros pasos en el uso de software educativo como material didáctico con el enfoque constructivista fueron dados por Papert (1999), el cual desarrolla una línea de software que corresponde a los lenguajes para el aprendizaje y de ella surge el lenguaje LOGO, que a partir de su desarrollo en el Instituto Tecnológico de Massachusetts (MIT) fue y es utilizado en numerosas escuelas y universidades en un sentido constructivista del aprendizaje.

Existen técnicas de enseñanza que se enmarcan en el enfoque constructivista asistido por computadoras, como son los ambientes de modelación, los que permiten que el aprendiz construya modelos. Compartimos el criterio de B. Moor and F. Deek (2014) , cuando consideran que las habilidades necesarias para el diseño, evaluación y construcción de un modelo incluyen la de descomponer un problema en sub-problemas más fáciles de resolver, integrar estas soluciones de forma coherente, reconocer la relación entre el modelo y el objeto original, para evaluar el modelo y cambiarlo si es necesario, re-usando (característica de la programación) el código que sea necesario.

Consideramos que en el aprendizaje de la programación, es necesario partir de la modelación de un proceso real, para ello se van aprendiendo diferentes esquemas lógicos para diseñar algoritmos que resuelvan problemas similares, estos esquemas con el tiempo se incorporan automáticamente en el proceso de diseño como operaciones mentales, en la medida en que se van modelando problemas más complejos se añaden nuevos conceptos que modifican los esquemas anteriores, como es el caso de la eficiencia computacional. En los dos primeros semestres de la carrera se enseña a diseñar algoritmos y a implementarlos en un lenguaje de programación, primero estructurado y después Orientado a Objetos, el objetivo es que el programa resuelva el problema propuesto. En el tercer y cuarto semestres se introduce una asignatura encargada de enseñar a programar con eficiencia, esto añade nuevos conceptos y maneras de enfocar la programación y es necesario reorganizar los

nuevos esquemas y crear nuevas estructuras mentales, los nuevos conceptos se vuelven más generales e incluyen los anteriores. Este proceso no siempre es respetado en la enseñanza de la programación de computadoras, como ya se ha expuesto no se tienen en cuenta las condiciones psicológicas, biológicas ni pedagógicas; no se respeta el ritmo del aprendizaje y el alumno no tiene ocasión de organizar, asimilar y acomodar lo aprendido para usarlo en la solución de nuevos y más complejos problemas del mundo real. Consideramos, entonces, que el estudio y aplicación del enfoque psicogenético de Piaget en la enseñanza de la Programación es un fundamento importante para los docentes de esta disciplina, que ayuda a diseñar nuevos ambientes y objetos de aprendizaje basados en tecnología que respondan a las necesidades individuales de cada estudiante y les permita lograr un equilibrio entre los acontecimientos externos y sus propios esquemas (Piaget, 1971, 1978).

Nos suscribimos al enfoque de Piaget, para nosotros novedoso, después de haber incursionado en el uso de herramientas de visualización y organización del conocimiento como son los mapas conceptuales introducidos por Joseph Novak and Gowin (1988), basándose en el aprendizaje como procesamiento de información, en respuesta a la línea de Ausubel del aprendizaje significativo dentro del marco de un programa denominado "Aprender a Aprender". Consideramos que estos enfoques también son válidos en la enseñanza de la programación de computadoras ya que, en ellos, el conocimiento está organizado y representado en todos los niveles de abstracción.

Conclusiones

Las herramientas de visualización de programas contribuyen al aprendizaje de las disciplinas de programación de computadoras, uno de los problemas más acuciantes de la carrera Ingeniería en Sistemas computacionales.

El enfoque constructivista, la enseñanza significativa y el enfoque psicogenético de Piaget, deben ser fundamentos a considerar para diseñar ambientes mediados por tecnologías que permitirán desarrollar aplicaciones educativas eficaces en la enseñanza de la Programación de Computadoras, facilitando así la colaboración e intercambio entre los sujetos del proceso; así mismo se determina que el enfoque de Piaget es muy útil para diagnosticar lo que el alumno ya sabe y cómo ir, paulatinamente, usando sus potencialidades, incorporando los nuevos conceptos siempre en un mayor nivel de complejidad, respetando su ritmo para que pueda asimilar, acomodar y aplicar lo aprendido, desarrollando nuevas estructuras mentales.

Bibliografía

ACM, & IEEE-CS. (2014). Computing Curricula 2009. The Overview Report covering undergraduate degree programs in Computer Engineering, Computer Science, Information Systems Information Technology, Software Engineering (pp. 58): ACM & IEEE-CS.

Almeida, F., Blanco, V., & Moreno, L. (2013). *EDApplets: Una Herramienta Web para la Enseñanza de Estructuras de Datos y Técnicas Algorítmitcas.* Paper presented at the X Jornadas de Enseñanza Universitaria de la Informática, Universidad de Laguna. Tenerife.

Ausubel, D., Novak, J., & Hainesian, H. (2000). *Psicología Educativa. Un punto de vista cognocitivo* (Trillas ed. Vol. 1). México.

Barbe, W. B., & Milone, M. N. (2015). What we know about modality strengths. *Educational Leadership, 38*, 378-380.

Barnes, D., & Kolling, M. (2015). *Objects first with Java: A practical introduction to using BlueJ* (Prentice Hall/Pearson Education ed. Vol. 1). Harlow, England.

Bennedsen, J., & Caspersen, M. (2014). Failure rates in introductory programming. *ACM SIGCSE Bulletin, 39*(2), 45-67.

Bladek, C., & Deek, F. P. (2010). Understanding novice programmers difficulties as a requirement to specifying effective learning environments. *New directions in higher education, Nova Science,* 5.

Cañas, A., Granados, A., Pérez, C., Pérez, J., & Hill, G. (2015). The network architecture of CmapTools (Technical Report No. IHMC CmapTools 2003-01). , FL: Institute for Human and Machine Cognition. *Pensacola*.

Cañas, A., & Novak, J. (2004). *Concept Maps: Theory, Methodology, Technology.* Paper presented at the First Int. Conference on Concept Mapping, Spain.

Castillo, J., & Barberán, O. (2016). Mapas Conceptuales en Matemáticas. Retrieved 12-3-2018, 2018, from http://www.cip.es/netdidactica/articulos/mapas.htm

Cooper, S., Dann, W., & Pausch, R. (2012). *ALICE: A 3-D tool for introductory programming concepts.* Paper presented at the 5th Annual CCSC Northeastern Conference, New York.

Chestlevar, C. I. (2015). Utilización de Mapas Conceptuales en la enseñanza de la programación. *Informática Aplicada, 2*, 11.

Deek, F. P. (1997). *An integrated environment for problem solving and problem development.* (Ph.D. Computer and Information Science), Institute of Technology, Newark, New Jersey.

Deek, F. P., & McHugh, J. (2013). SOLVEIT: An experimental environment for problem solving and program development. *Journal of Applied Systems Studies, 2*(2), 376-396.

Deek, F. P., McHugh, J., & Hiltz, S. R. (2013). Methodology and technology for learning programming. *Systems and Information Technology, 4*(1), 25-37.

George, C. (2011). *Experiences with novices: The importance of graphical representations in supporting mental models.* Paper presented at the 12th Annual Workshop of the Psychology of Programming Interest Group.

Good, J., & Brna, P. (2014). *Novice difficulties with recursion: Do graphical representations hold the solution?* Paper presented at the European Conference on Artificial Intelligence in Education, Lisbon, Portugal.

Guttag, J., & Liskov, B. (1986). *Abstraction and Specification in Program Development.* Leipzig: The MIT Press.

Guzdial, M., & Elliott, A. (2015). *Imagineering inauthentic legitimate peripheral participation: an instructional design approach for motivating computing education.* Paper presented at the International workshop on Computing education research, Canterbury, United Kingdom.

Hennessy, S. (2013). Learner perceptions of realism and magic in computer simulations. *British Journal of Educational Technology, 24*.

Hung, T., & Rodger, S. (2016). *Increasing Visualization and Interaction in the Automata Theory Course.* Paper presented at the 41st SIGCSE Technical Symposium on Computer Science Education, Austin, Texas.

Lezcano, M. (1998). *"Ambientes de aprendizaje por descubrimiento para la disciplina Inteligencia Artificial".* (Doctorado en Computación y Automática), Universidad Central "Marta Abreu" de Las Villas, Santa Clara.

Liffick, B., & Aiken, R. (2011). *A novice programmer's support environment.* Paper presented at the Integrating Technology into Computer Science Education.

Merril, P. (2016). *Computers in education* (Allyn & Bacon ed. Vol. 1).

Moor, B., & Deek, F. (2014). On the Design and Development of a UML-Based Visual Environment for Novice Programmers. College of Computing Sciences, New Jersey Institute of Technology, Newark, NJ, USA. *Journal of Information Technology Education, 5*, 1-24.

Moor, B., & Deek, F. (2014). *A Visualization Tool Using UML Subset to Aid the Novice Programmer.* Paper presented at the World Conference on Educational Multimedia, Hypermedia and Telecommunications 2005, Chesapeake.

Morris, J. (2003). Algorithm Animation: Using algorithm code to drive an animation. *Journal of Visual Langiuages and Computing, 2*(3), 6-15.

Muthukumarasamy, J., & Stasko, J. (2012). Visualizing Program. Executions on Large Data Sets using Semantic Zooming (pp. 17). Atlanta: Graphics,Visualization and Usability Center, CollegeofComputing, GeorgiaInstituteofTechnology

Novak, J. (1991). Ayudar a los alumnos a aprender cómo aprender. La opinión de un profesor-investigador" en Enseñanza de las Ciencias. *9,3*, 215-227.

Novak, J., & Gowin, D. (1988). *Aprendiendo a aprender* (Martínez Roca ed.). Barcelona.

Ontoria, A. (2014). *Mapas conceptuales: una técnica para aprender* (Narcea S.A. ed.). Javeriana.

Pane, J., & Myers, B. (2013). Usability issues in the design of novice programming systems (School of Computer Science Technical. Carnegie Mellon University ed., pp. 44). Pittsburgh, PA.

Papert, S. (1999). *¿Qué es Logo? ¿Quién lo necesita? Logo Philosophy and Implementation*: LCSI.

Piaget, J. (1971). *Psicología de la inteligencia*. Buenos Aires: Psique.

Piaget, J. (1978). *La equilibración de las estructuras cognitivas. Problema central del desarrollo*. Madrid: Siglo XXI.

Roman, G. C., & Cox, K. C. (1993). A taxonomy of program visualization systems. Visualización de Software Orientada a Comprensióon de Programas. *Computer, 26*(12), 11-24.

Satratzemi, M., Dagdilelis, V., & Evageledis, G. (2014). *A system for program visualization and problem-solving path assessment of novice programmers.* Paper presented at the Annual Joint Conference Integrating Technology into Computer Science Education, 6th Annual Conference on Innovation and Technology in Computer Science Education.

Señas, P., & Moroni, N. (2011). *Herramientas no convencionales para la enseñanza de la programación.* (Doctor en Ciencias Doctorado en Computación), Instituto de Ciencias e Ingeniería de Computación Universidad Nacional del Sur, Bahía Blanca-Argentina.

Soler, Y. (2009). *Aplicación de la visualización dinámica de programas en el diseño de estructuras de datos y el análisis de la complejidad de algoritmos.* (Doctor en Ciencias Técnicas), Universidad Central "Marta Abreu" de Las Villas, Santa Clara.

Stasko, J., Domingue, J., Brown, M., & Price, B. (2012). *Software Visualization: Programming as a Multimedia Experience.*: MIT Press.

Stasko, J., & Lawrence, A. (2007). Empirically Assessing Algorithm Animations as Learning Aids. *MIT Press*, 25.

Suchan, W., & Smith, T. (2012). *Using Ada as a tool to teach problem solving to non-CS majors.* Paper presented at the Annual International Conference on Ada.

Ziegler, U., & Crews, T. (2004). *An integrated program development tool for teaching and learning how to program.* Paper presented at the Technical Symposium on Computer Science Education, Portland.