MÉTODO DE ENSEÑANZA DE ALGORITMOS CENTRADO EN 2 DIMENSIONES.

Nieva García, Omar Santiago; Arellano Pimentel, J. Jesús*

RESUMEN

Este documento presenta un método de enseñanza de algoritmos para lograr el aprendizaje significativo en estudiantes que toman cursos introductorios a las ciencias computacionales. Nuestro trabajo se centra en dos dimensiones importantes del aprendizaje, por un lado la capacidad de abstracción y por otro la capacidad de resolución de problemas. Para poder desarrollar ambas dimensiones, se hace uso del paradigma de enseñanza constructivista y de la heurística de resolución de problemas matemáticos de Polya. De esa manera proponemos un esquema de aprendizaje que aquí describimos, así como un prototipo de software para apoyar la metodología propuesta.

INTRODUCCIÓN

El estudio de diversas carreras relacionadas con las ciencias computacionales implica el aprendizaje de importantes temas en áreas como sistemas de información, arquitectura de computadoras, redes, administración de tecnología y programación. De manera particular, el área de programación reviste una especial importancia dado que el diseño y construcción de programas (software) es el elemento principal para ofrecer soluciones mediante el uso de sistemas de cómputo. Para iniciarse en el aprendizaje de cualquier paradigma de programación, es un requisito tener habilidades en el desarrollo de algoritmos.

En planes y programas de estudio propuestos por la ACM[1], se incluye el estudio de los algoritmos como parte del área de Fundamentos de Programación. Así también, el Sistema de Universidades Estatales de Oaxaca (SUNEO) aborda este tema como materia obligatoria dentro de los cursos de perfil ingenieríl. La enseñanza de los algoritmos busca como principal objetivo desarrollar en el estudiante la habilidad de resolver problemas computables desde una perspectiva algorítmica, es decir, que el estudiante sea capaz de ofrecer una solución a través de una secuencia de pasos finitos y libres de ambigüedad.

Existe actualmente una gran cantidad de material disponible para la enseñanza de algoritmos. Sin embargo consideramos que se ha fallado en lograr un aprendizaje significativo[2] en los estudiantes debido a que este material no es homogéneo y abarca solo algunos aspectos relevantes en la enseñanza. El trabajo de Buck y Stucki[3] es una referencia importante para afirmar lo anterior. En éste se subraya que aunque es obvio para muchos educadores que los estudiantes deberían dominar los fundamentos antes de intentar conocimientos más avanzados; el ímpetu del avance tecnológico en el área de computación ha perdido de vista este importante aspecto. Es decir, actualmente se privilegia más la información de carácter técnico y no el uso de una metodología para desarrollar y mejorar las habilidades básicas dentro del área de programación.

Otro estudio que tiene relación con lo descrito en la cita anterior es el de Baeza-Yates[4] quien hace una revisión de los principales libros orientados hacia algoritmos, enfatizando que los libros de carácter introductorio están asociados hacia lenguajes de programación específicos y en su mayoría no orientados hacia los algoritmos per se.

* Universidad del Istmo

Otra tipo de propuestas orientadas hacia la enseñanza de los algoritmos son las herramientas de software educativo. Algunos trabajos de investigación han derivado en herramientas como Synergo[5], Raptor[6], Jgrasp[7], JkarelRobot[8] y Jeliot[9], sin embargo solo se utilizan parcialmente, debido a que estas herramientas sólo pueden aprovecharse una vez que el alumno ha entendido como escribir un algoritmo y no en el proceso de enseñanza de conceptos básicos como lo son las expresiones lógicas, expresiones de asignación o el ámbito de variables y constantes.

En este documento proponemos una metodología que hemos llamado *Enseñanza de algoritmos centrado en 2 dimensiones*. Esta metodología busca que al estudiante desarrolle su capacidad de abstracción (no solamente enseñarle a escribir algoritmos) en la resolución de diversos problemas mediante soluciones algorítmicas. Así también busca avanzar progresivamente en sus niveles de conocimiento para facilitar el uso de conceptos cada vez más complejos. El desarrollo y contenido de esta metodología se detalla en un libro que actualmente se elabora y en un prototipo de software que servirá como herramienta básica de apoyo a nuestra propuesta.

TRABAJO RELACIONADO

La enseñanza de algoritmos es un tema que ha sido estudiado por varios años. Sin embargo, lograr que ésta enseñanza se refleje en un aprendizaje significativo[2] del estudiante presenta varios retos. Para discutir el trabajo relacionado en esta sección, lo hemos clasificado en dos categorías: estrategias de enseñanza de algoritmos y, por otro lado, herramientas visuales para la enseñanza de algoritmos.

Se han desarrollado diversos estudios tendientes a mostrar las mejores prácticas para la enseñanza de algoritmos, debido a la importancia que reviste para la correcta formación de habilidades en ésta área. Uno de estos primeros estudios es el de Baeza-Yates[4], donde se muestra toda la secuencia de materias relacionadas con el área de programación y en donde se enfatiza el orden que deberían seguir estas materias, comenzando por un curso de introducción a la programación con una orientación algorítmica. El estudio hace también una revisión de los principales libros orientados hacia algoritmos, enfatizando que los libros de carácter introductorio están asociados en su mayoría hacia lenguajes de programación y no orientados hacia los algoritmos.

Un artículo que subraya de manera importante la aplicación de aspectos cognositivos en la enseñanza de la programación es el de Buck y Stucki[3]. Este artículo muestra la relación que existe entre la enseñanza de diferentes conceptos de programación con los niveles de una jerarquía de conocimiento llamada taxonomía de Bloom. Señala que en el proceso de enseñanza, actualmente los educadores del área de ciencias computacionales han perdido de vista que los estudiantes deben desarrollar habilidades básicas de abstracción antes de intentar tareas más avanzadas. Se indica que existen 6 niveles de conocimiento, según la taxonomía de Bloom, y que la habilidad de programar se logra hasta un 5o. nivel (síntesis); sin embargo, añade, hoy en día se pide a los estudiantes programar sin haber pasado por los 4 niveles de conocimiento previos.

Un trabajo alterno, que aunque enfocado a algoritmos, examina la posibilidad de colaboración para el diseño de algoritmos es el de Voyiatzaki[5]. Su propuesta de enseñanza se basa en un enfoque colaborativo, de forma que grupos de estudiantes puedan interactuar sobre un mismo diagrama de flujo o pseudocódigo.

En nuestro país, no encontramos estudios formales que profundicen en los métodos y mejoras en la enseñanza de los algoritmos a nivel universitario.

Por lo que respecta al desarrollo de herramientas visuales para la enseñanza de algoritmos, estas buscan completar las técnicas tradicionales de desarrollo de pseudocódigo y diagramas de flujo en el aula. Las primeras herramientas propuestas basan su funcionamiento en los denominados Diagramas de Estructura de Control (CSD) y se vinculan a algún lenguaje de programación. Por ejemplo GRASP, propuesto por Cross et. al.[10], presentan esta herramienta como un recurso para el análisis de estructuras secuenciales, de decisión y de control en el lenguaje Pascal y ADA. Esta herramienta ha evolucionado y en un artículo posterior, presenta un entorno de desarrollo con una versión mejorada de la herramienta antes descrita, bajo el nuevo nombre de JGRASP[7]. La versión actualizada centra su atención en la visualización de estructura de datos a partir de código escrito en Java. Un dato importante a resaltar, es que los autores señalan que aunque se ha demostrado que las técnicas de visualización son pedagógicamente efectivas, no hay tantas implementaciones de éstas como se quisiera.

Un caso de estudio que se deriva de la aplicación de las taxonomías de Bloom a la enseñanza de la programación, es el artículo desarrollado por Buck y Stucki[8] donde se presenta la herramienta JkarelRobot. Esta herramienta esta desarrollada para soportar lenguaje Java, Pascal y Lisp y enfatiza el hecho de centrarse en conceptos de programación y no en la sintaxis.

El artículo de Giordano y Carlisle[6], señala que una herramienta visual mejora el desempeño de los estudiantes y facilita la construcción de programas. La herramienta propuesta se denomina RAPTOR y es comparada con otras herramientas (algunas de ellas comerciales) como BACCII, SFC, B# y ALICE; lo cual permite darnos una idea más general de la evolución y características de estas herramientas. Un aspecto importante a resaltar de este artículo, es que nuevamente se hace referencia a la taxonomía de Bloom, señalando que el alcance de esta herramienta es hasta el nivel 3 de dicha clasificación.

Otra herramienta que destaca en la investigación preliminar es Jeliot[9], que a partir de código de alto nivel realiza animaciones de los principales elementos de dicho código como; tipos de datos, estructuras de datos, operaciones de asignación, estructuras de control, entre otras.

Finalmente, una de las deficiencias que se quiere atacar en éste estudio, es el de evitar que el estudiante tenga como hábito el ensayo-error al momento de desarrollar un programa. En relación a este aspecto encontramos en el artículo de Edwards[11], que aunque no es una herramienta propiamente dicha, detalla el desarrollo de pruebas dirigidas (TDD) como una actividad que debería incluirse en las herramientas de enseñanza de programación, a fin de que el estudiante se acostumbre a la detección temprana de errores, el desarrollo incremental y reafirme su autoconfianza al ir generando en cada etapa código correcto.

Como observación adicional, cabe destacar el hecho de que casi todas las herramientas de visualización son usadas en cursos de programación y no en cursos introductorios de algoritmos, abriéndose una oportunidad para explotar ahí dichas herramientas.

MARCO TEÓRICO

Algoritmos.

Algoritmo, según la Real Academia de la Lengua Española, es un conjunto ordenado y finito de operaciones que permiten encontrar la solución a un problema. Ejemplos de algoritmos sencillos son una receta de cocina o las instrucciones para armar un juguete. Los primeros algoritmos registrados fueron originados en el área matemática, como métodos para resolver un problema usando una secuencia de cálculos más simples.

El desarrollo de un algoritmo es un proceso de varias etapas. Se parte de un problema que es necesario resolver. Se diseña una solución. Se prueba si dicha solución cumple con los requisitos de completes (contempla todos los casos posibles) y correctes (no contempla casos inapropiados en sus soluciones). Finalmente, si la solución cumple con dichos requisitos, se codifica usando algún lenguaje de programación.

El planteamiento o definición de un problema debe permitir tener una visión general del mismo, estableciendo las condiciones iniciales y sus límites (donde empieza y termina). Esta etapa se enfoca en identificar los resultados que se esperan de la solución, identificar los datos disponibles o entradas necesarias a partir del problema dado y establecer un mecanismo o proceso para transformar tanto los datos disponibles como las entradas necesarias en los resultados deseados.

La etapa de diseño proporcionan directrices generales para resolver problemas bajo un enfoque algorítmico. Estas técnicas pueden clasificarse según Levitin[12] en:

- Fuerza Bruta. En la práctica es una de las más usadas, y aunque no conduce a algoritmos óptimos, pueda aplicarse a un sin fin de problemas.
- Divide y vencerás. Técnica basada en la partición de un problema en subproblemas más pequeños; del mismo tamaño y tipo; se resuelven los subproblemas y sus soluciones se combinan para obtener la solución del problema original.
- Disminuye y vencerás. Consiste en resolver un problema, reduciendo el tamaño de las instancias lo más posible y luego resolver recursivamente hasta obtener la solución de la instancia original.
- Transforma y vencerás. Se basa en transformar una instancia dada en otra para un mismo problema, facilitando de esta manera su solución.

De acuerdo a lo anterior, como parte de la enseñanza inicial de algoritmos se necesitan considerar técnicas generales de diseño. Posteriormente, cuando la complejidad computacional se vuelve importante; serán necesarios temas adicionales como matemáticas discretas, teoría de autómatas y lenguajes formales.

Las pruebas de la solución obtenida, permiten simular el funcionamiento del algoritmo en función de los datos iniciales del problema y de datos similares que permitan prever posibles escenarios. Si los resultados son correctos y las posibles circunstancias en que puede funcionar el algoritmo son contempladas, se podrá seguir adelante. En caso contrario se deberán hacer adecuaciones al algoritmo y repetir los pasos anteriores hasta que la solución sea correcta y completa.

Finalmente la codificación en algún lenguaje de programación, implica convertir las instrucciones del algoritmo en instrucciones de computadora.

Modelos de Enseñanza.

La enseñanza consiste, de forma general, en comunicar conocimiento, habilidades o experiencias a alguien para su aprendizaje, empleando para ello métodos y técnicas.

Los métodos o modelos de enseñanza están relacionados con la forma en cómo se enseña nuevo conocimiento a un estudiante. Existen tres modelos predominantes[13] en este sentido; transmitivo, de condicionamiento y constructivista. Cada uno busca la mejor manera de enfocar la enseñanza, transmitirla y evaluarla. A partir de estos tres elementos, podemos entonces abordar y describir algún modelo de enseñanza en general.

En el modelo transmitivo o tradicional, se concibe la enseñanza como aquella en la que el profesor toma el papel principal exponiendo conceptos y el alumno toma una actitud pasiva recibiendo dichos conceptos. Es decir, se limita a la comunicación entre un emisor y un receptor, haciendo a un lado el fenómeno de comprensión y de uso de los sentidos.

El modelo condicionamiento o conductista hace uso de refuerzos apropiados para lograr un comportamiento terminal deseado y reducir las respuestas no deseables, por ejemplo, los maestros pueden mejorar el comportamiento de los estudiantes por medio de recompensas sistemáticas para aquellos que sigan las reglas.

Finalmente, el modelo constructivista o perspectiva radical, concibe la enseñanza como una actividad crítica y al docente como un profesional autónomo que constantemente investiga y mejora su práctica. En este sentido, la enseñanza no es solamente la transmisión de conocimientos, es la innovación de métodos de apoyo que permitan a los estudiantes construir su propio saber; es decir, cada individuo aprende construyendo su propia estructura cognoscitiva. Se reconoce a tres autores como los más destacados de esta corriente Vygotski, Piaget y Ausubel.

Precisamente, Ausebel es el autor del concepto aprendizaje significativo[2], al que se ha recurrido sistemáticamente por varias décadas para enfatizar que el estudiante realmente puede aprender. Se trata de una teoría constructivista basada en la premisa de que el estudiante es quien genera y construye su aprendizaje. Algunos de los principios de esta teoría son los siguientes[14]:

- El nuevo conocimiento debe estar relacionado con la experiencia y el contexto social de quien aprende.
- La adquisición de nuevo conocimiento demanda una actitud activa por parte del estudiante y de que éste se involucre en actividades de aprendizaje, que le permitan continuar aprendiendo por si mismo.
- Usar el conocimiento previo para aprender. El aprendizaje de conceptos altamente abstractos no es posible sin tener estructuras de conocimiento desarrolladas previamente.

De esta manera el aprendizaje significativo busca ser permanente (aprendizaje a largo plazo), produce conocimientos (se pasa de no saber a saber) y esta basado en la experiencia (depende de conocimientos previos).

El modelo constructivista, promueve la organización de las materias dentro de un plan de estudios en espiral, de forma que el estudiante construya nuevo conocimiento a partir del aprendido previamente. Aunque dicho modelo se ha aplicado más en áreas de pedagogía y matemáticas [14], existen algunas aplicaciones en el área de Ciencias de la Computación. Seymour Papert[15], un connotado matemático y educador, afirma que es más fácil que alguien aprenda cuando se le pide construir un producto, algo externo a sí mismo. De esto deduce que el uso de una computadora en educación bajo criterios de construcción por parte del alumno, es decir como instrumento didáctico, fomenta el aprendizaje.

Software Educativo.

El software educativo involucra la utilización de las nuevas tecnologías en el proceso de enseñanza aprendizaje y concretamente del uso de la computadora como una herramienta y como medio a través del cual se puede aprender significativamente[16].

Por lo general se tiende a pensar que el software educativo es cualquier programa de computadora que permite cumplir o apoyar funciones educativas. Sin embargo, las tendencias actuales son más restrictivas a la hora de clasificar un programa de computadora como software educativo; solo se consideran aquellos programas diseñados específicamente con fines didácticos que permitan el desarrollo de habilidades cognitivas. Ello se debe en parte a que los objetivos que se persiguen a la hora de desarrollar software educativo difieren sustancialmente de los objetivos perseguidos al desarrollar cualquier otro tipo de software.

Existen diversas clasificaciones de software educativo. Thomas Dwyer[17] propone una taxonomía en función del enfoque educativo predominante en: algorítmico o heurístico. En un software de tipo algorítmico predomina el aprendizaje vía transmisión de conocimientos, así el diseño del software ha de encapsular secuencias de actividades de aprendizaje que guíen al alumno a su propio ritmo. El rol del alumno es asimilar al máximo lo que se intenta transmitir.

En un software de tipo heurístico predomina el aprendizaje por experiencia y descubrimiento, el diseño del software debe enfocarse hacia ambientes con abundantes situaciones que el alumno pueda explorar y/o generar. Así el alumno debe llegar al conocimiento a través de su experiencia.

Tomando como base la taxonomía antes mencionada, una clasificación más que puede hacerse esta en base a las propias funciones que el software realiza. Dentro de los algorítmicos se tienen: sistemas tutoriales, sistemas de ejercitación y práctica, entre otros. Dentro de los heurísticos están: juegos educativos, simuladores, micromundos, sistemas expertos, entre otros.

La incorporación de un software educativo como apoyo didáctico plantea dos caminos a seguir: elegir un software existente, o construir uno a la medida. Elegir y validar el software más apropiado en función de los contenidos curriculares no es una tarea fácil. Existen estudios a profundidad que plantean como debe hacerse una elección y validación apropiada[18]. Sin embargo no siempre es posible hallar un software que se adapte perfectamente a lo que se desea.

Para desarrollar software educativo existen diversas metodologías que establecen una forma particular de como hacerlo[17]. La mayoría de estas metodologías parten del análisis de las necesidades educativas o curriculares, posteriormente se realizan las fases de diseño, implementación, prueba piloto, pruebas de campo y retroalimentación. Algunos de los aspectos a considerar al diseñar software educativo que no se consideran en otro tipo de software son: qué enseñar, con qué metodología, cómo saber que el aprendizaje se está logrando, cómo motivar y mantener motivados a los alumnos, qué funciones de apoyo debe tener el software, cuáles componentes son los más apropiados en la interfaz de usuario en función de los contenidos educativos, en qué momento y cuál información se ha de presentar al usuario, entre otros.

PROPUESTA METODOLÓGICA

Esta sección presenta nuestra propuesta denominada *Enseñanza de Algoritmos centrada en 2 dimensiones.* Estas dimensiones buscan mejorar la enseñanza de los algoritmos en dos aspectos que consideramos no han sido cubiertos por otras propuestas. Por un lado, mejorar la capacidad para resolver problemas y por otro lado mejorar la capacidad de abstracción (ver Figura 1).

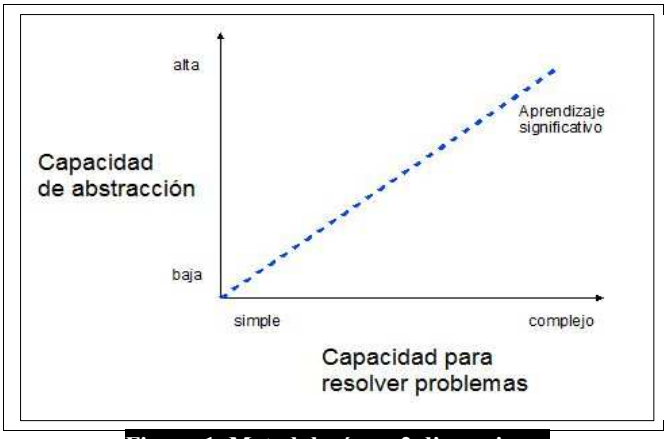


Figura 1. Metodología en 2 dimensiones

La resolución problemas implica desarrollar por parte del estudiante la habilidad para leer un texto, interpretarlo y determinar la incógnita o incógnitas que una situación plantea. Dado que los algoritmos son una secuencia de pasos para resolver un problema determinado, solamente se podrá llegar a esta solución si el estudiante es capaz de delimitar y entender el problema.

La capacidad de abstracción permite aislar los detalles de un objeto o situación dada y concentrarse en los aspectos más relevantes de ésta. En la programación de cualquier dispositivo electrónico (computadoras, pda's, teléfonos celulares) lo importante es determinar que aspectos son relevantes para la solución de un problema, en lugar de perderse en los detalles de marcas, modelos o plataformas.

Estas dos dimensiones; capacidad para resolver problemas y capacidad de abstracción, son abordadas mediante la *Heurística de Resolución de Problemas* propuesto por G. Polya[19] y el uso de la *Taxonomía de los objetivos educativos* propuesta por Bloom[20].

En el caso de la heurística propuesta por Polya, consiste en una serie de pasos y preguntas tendientes a ir paulatinamente analizando el texto de un problema, con la

finalidad de encontrar una solución al mismo. El proceso de esta heurística se puede comprender mediante la Tabla 1.

Tabla 1. HEURÍSTICA DE RESOLUCIÓN DE PROBLEMAS

Etapa	Descripción
Entender del problema	Es necesario entender el problema. Es posible formular preguntas como ¿Cual es la incógnita? ¿Cuales son los datos disponibles? ¿Existe alguna condición para resolver la incógnita? Aquí es posible hacer uso de esquemas, anotaciones o dibujos que faciliten la compresión del problema.
Obtener un plan de solución	Idear una estrategia que permita llegar de los datos disponibles hasta la resolución de la incógnita. Es posible formular preguntas como ¿Es un problema similar a otro ya visto previamente? ¿Existe algún método que use los datos proporcionados y nos devuelva un resultado adecuado? ¿De que otra forma se puede plantear el problema?
Aplicar el plan de solución	Llevar a cabo la estrategia sugerida y verificar cada paso de ésta. Es posible formular aquí preguntas como ¿Cada paso es evidente y claro?
Revisar la solución	Analizar la solución obtenida, a fin de estar seguros que es correcta y satisfactoria. Es posible formular aquí preguntas como ¿El resultado es el que se esperaba? ¿El resultado se puede obtener mediante otro método? ¿El método usado se puede aplicar a otra clase de problemas?

En la Figura 2 se esquematizan las cuatro etapas de la heurística de Polya. Como se pude observar, se trata de un proceso iterativo, que puede avanzar o regresar entre cada una de las actividades propuestas.

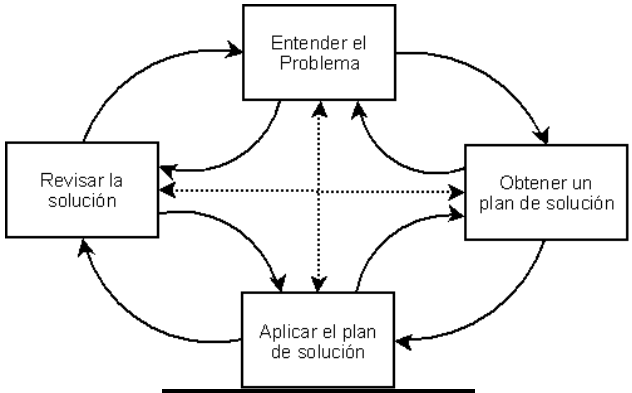


Figura 2. Heurística de Polya

Por lo que respecta a la taxonomía propuesta por Bloom, un resumen de esta se presenta en la Tabla 2.

Tabla 2. TAXONOMÍA DE BLOOM

Nivel	Descripción	Actividades Enfocadas a Algoritmos
Conocimiento	Se refiere a la capacidad del estudiante de recordar conceptos previamente aprendidos. A partir de elementos específicos es posible completar conceptos	- Reconocer la sintaxis de una instrucción. - Reconocer la semántica de una instrucción.
Comprensión	Es la capacidad de captar el significado de de un texto, pudiéndolo traducir a una representación específica. Capacidad de llevar a cabo resúmenes o predecir consecuencias o efectos.	- Predecir los posibles efectos al ejecutar un algoritmo línea por línea. - Predecir que cambios ocurren al modificar una instrucción. - Traducir un algoritmo dado a pseudocódigo o diagrama de flujo.
Aplicación	Es la habilidad para usar conceptos aprendidos en una situación nueva y concreta. Los conceptos abarcan reglas, métodos, principios, leyes o teorías.	- Usar de funciones de librería. - Usar estructuras con un fin determinado y claro (decisiones, ciclos). - Crear rutinas para llevar a cabo una tarea específica.
Análisis	Es la habilidad para descomponer un todo en partes y así entender como esta organizado. Abarca la capacidad de identificar partes, sus relaciones, y como colaboran las partes para llegar a un objetivo común.	- Leer un algoritmo y hacerle cambios. - Ubicar posibles errores en un algoritmo.
Síntesis	Se refiere a la capacidad de reunir partes para formar un componente, poniendo énfasis en la creatividad, para dar origen a nuevos modelos o soluciones.	- Construir un algoritmo completo a partir de las especificaciones de un problema.
Evaluación	Es la capacidad para juzgar la relevancia de un material (libro, reporte, tesis) dado un propósito específico. Para ello se hace necesario establecer criterios.	- Evaluar requerimientos a nivel de usuario, sistema, organización y desarrollar una solución algorítmica coherente que los tome en cuenta. - Compara soluciones algorítmicas que resuelven un mismo problema.

Las etapas de este proceso pueden verse como una estructura piramidal, donde el estudiante va adquiriendo nuevas habilidades y conocimientos conforme asciende de nivel (ver Figura 3). De esta manera se presta atención a un aspecto que en muchos casos se pasa por alto en la enseñanza de algoritmos, el hecho de que el estudiante avanza gradualmente desde un nivel de *conocimientos conceptual* (parte más baja de la pirámide) hasta que puede ser capaz de *evaluar* diferentes requerimientos y cubrirlos mediante una solución integral.

Figura 3. Taxonomía de Bloom

Con los elementos anteriores, pretendemos garantizar un aprendizaje significativo en los estudiantes que lleven cursos de algoritmos o de introducción a la programación. Por un lado, el eje relacionado con la *Capacidad de Resolver Problemas*, pretende que el estudiante comience por trabajar y solucionar problemas muy simples hasta llegar a problemas de alta complejidad. Por otro lado, el eje relacionado con la *Capacidad de Abstracción*, considera el incremento gradual del nivel de conocimientos, que permita al estudiante comenzar con bajos niveles de abstracción, hasta alcanzar altos niveles; que le permitan tener una visión completa del desarrollo de soluciones algorítmicas.

Lo anterior da como resultado un esquema de aprendizaje[21] que comprende tres artefactos principales:

- Recursos de aprendizaje
- Actividades de aprendizaje
- Apoyos de aprendizaje

Los recursos de aprendizaje proporcionan el contenido del curso y permiten al estudiante construir su conocimiento, generalmente se trata de libros, notas o tutoriales. Las actividades de aprendizaje establecen lo que el alumno debe hacer para aprender, de forma tradicional se refieren a la resolución de ejercicios y problemas, proyectos e investigaciones. Finalmente, los apoyos de aprendizaje brindan el soporte necesario para ayudar al alumno a cubrir sus necesidades individuales de aprendizaje, por lo regular se trata de software educativo.

En cada uno de esto artefactos, hemos colocado elementos que se obtienen a partir de las *dimensiones* antes señaladas. De forma que en la Figura 4, se pude observar la forma el esquema de aprendizaje que proponemos cubre cada uno de los artefactos señalados haciendo uso de las dimensiones identificadas previamente.

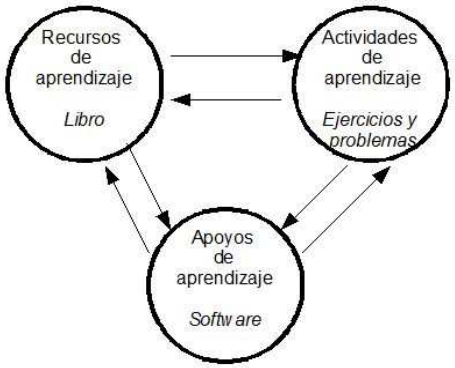


Figura 4. Artefactos y sus elementos

PROTOTIPO

Como principal apoyo didáctico de la metodología aquí descrita se propone un prototipo de software estructurado en base a las etapas de la heurística de Polya tal como se presenta en la Figura 5.

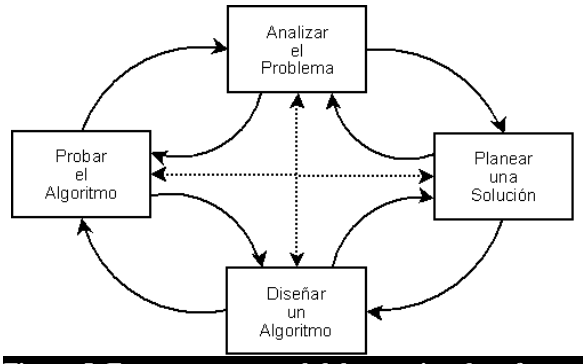


Figura 5. Estructura general del prototipo de software

En la etapa de análisis del problema el software debe permitir la escritura del enunciado inicial del problema, ya que a partir este enunciado inicial se plantean una serie de 8 preguntas al usuario, estas son:

1.- ¿Cuáles son los datos iniciales?
2.- ¿Qué es necesario preguntar para completar los datos iniciales?
3.- ¿De dónde se tomarán los datos iniciales?
4.- ¿Cuáles son los supuestos?
5.- ¿Cuál es la incógnita?
6.- ¿Qué es lo que se quiere resolver o calcular?
7.- ¿Qué información ha de presentarse como resultado?
8.- ¿A través de que forma se presentarán los resultados?

La primera pregunta se refiere a los datos que se dan en el enunciado del problema y que se usarán posteriormente tanto en la planeación como el diseño del algoritmo. De la respuesta a la segunda pregunta se ubicarán aquellas variables o datos de entrada necesarios para el resolver la o las incógnitas a través del algoritmo. La tercera pregunta va en el sentido de suponer o determinar si los datos serán introducidos por el usuario a través de un teclado, tomados desde un archivo de datos, o algún otro lugar. La respuesta a la cuarta pregunta ubica situaciones que serán establecidas como ciertas

aunque no se especifiquen en el enunciado original del problema, sin embargo dichos supuestos no deben alterar lo que el problema solicita resolver. La respuesta a la quinta pregunta puntualiza la incógnita o incógnitas que se pretenden encontrar o calcular, por lo general una incógnita esta relacionada con la información más relevante que la solución algorítmica debe entregar como resultado. La respuesta a la sexta pregunta debe plantear cómo calcular la incógnita o incógnitas a partir de los datos y variables de entrada identificados en la respuestas a las dos primeras preguntas, esta respuesta es de mucha importancia ya que es el primer bosquejo de solución en términos de lo que hasta ahora se identificado o inferido del enunciado del problema. La respuesta a la séptima pregunta pretende puntualizar explícitamente el enunciado que ha de acompañar al dato asociado a la incógnita o incógnitas resueltas. Por último la octava pregunta va en el sentido de si los resultados se han de entregar en pantalla, impresos, en archivo, etc. La Figura 6(a) presenta la pantalla propuesta asociada a la etapa de analizar el problema que debe de proveer el prototipo de software.

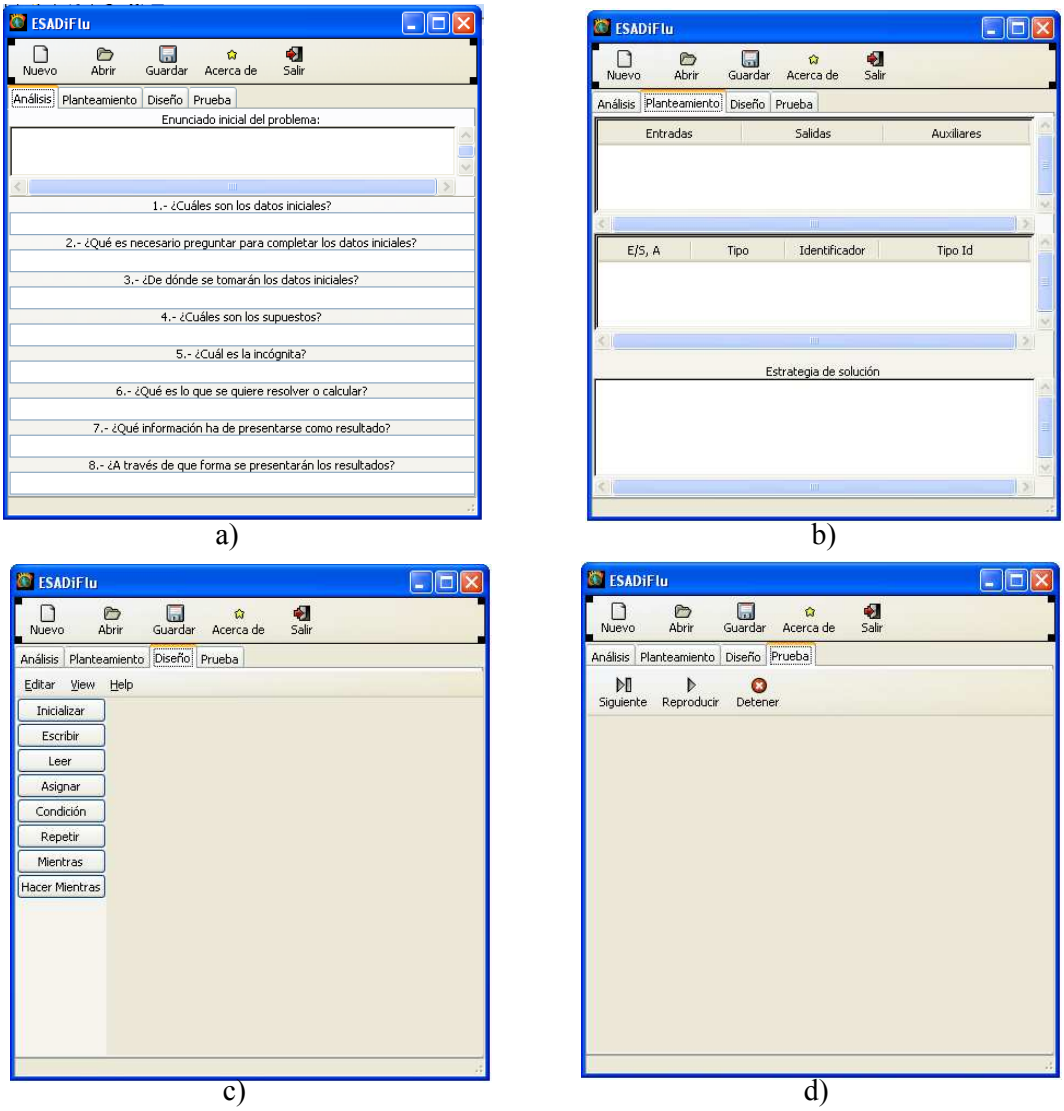
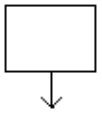
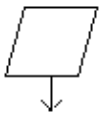
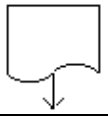


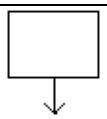
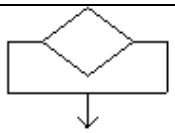
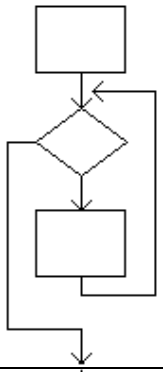
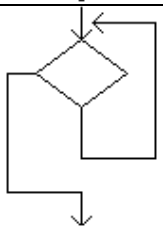
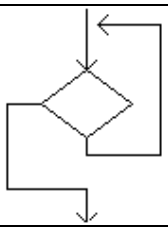
Figura 6. Las cuatro pantallas principales del prototipo de software propuesto

Concluida la primera etapa es posible planear una solución. En esta segunda etapa es necesario tabular las entradas, salidas y quizá los datos iniciales identificados en las respuestas a la segunda, quinta y primer pregunta respectivamente. Cabe la posibilidad de que también se hayan identificado algunas variables auxiliares en la respuesta de la sexta pregunta para almacenar cálculos intermedios antes de calcular alguna incógnita, en este caso también debe tabularse. El prototipo de software debe proveer los elementos necesarios para tabular las entradas, salidas y variables auxiliares, esto se puede visualizar en la Figura 6(b). La segunda tabla que debe generarse en esta etapa es aquella que permita tipificar y asociar un identificador a cada entrada, salida o variable auxiliar. La interfaz del prototipo debe facilitar la generación de dicha tabla rellenando automáticamente la primera columna y mediante una lista de selección el tipos permitir asociar uno de ellos (entero, real, carácter o lógico). Construidas las tablas anteriores se tiene la posibilidad de reescribir la respuesta a la sexta pregunta en términos de los identificadores recién creados y así planear con mayor formalidad la estrategia de solución del problema.

La etapa correspondiente a diseñar un algoritmo parte la conclusión de la etapa anterior, esto permite de alguna forma sistematizar la construcción de un diagrama de flujo para dar solución al problema. Esta sistematización se logra si damos por hecho que todo identificador asociado a una entrada en algún punto de nuestro diagrama debe ser leído, y todo identificador asociado con una salida en algún punto de nuestro diagrama debe ser escrito. Así también las incógnitas y variables auxiliares en algún punto del diagrama deben ser calculadas antes de presentarse o utilizarse como información en el algoritmo. En el caso de que ocurra que una entrada no es leída, o una salida no es escrita, o un identificador no es utilizado, el software debería mostrar un mensaje de aviso reportando esta anomalía ya que ello implica una falla en las etapas previas. Por otro lado, si al diseñar el algoritmo se considera que es conveniente contar con una variable auxiliar habrá que retroceder a la etapa anterior para tabularla y reflejar su uso en la estrategia de solución, de lo contrario el software debería emitir un mensaje de error para informar que se esta utilizando un identificador no considerado en el plan de solución. En esta etapa de diseño, el software debe proveer un conjunto de elementos que permitan la edición del diagrama de flujo, un resumen de estos elementos se presentan en la Tabla 3.

Tabla 3. ELEMENTOS DE UN DIAGRAMA DE FLUJO

Elemento	Propósito	Figura asociada	Restricciones de sintaxis
Inicializar	Inicializa todos los identificadores a utilizar. El primer valor asignado define el tipo de dato asociado al identificador.		Solo se pueden asignar datos más no operaciones de cálculo u otros identificadores
Escribir	Escribir un mensaje o dato a un dispositivo de salida, por lo general se trata de la pantalla.		Las identificadores a escribir deben estar tabulados como salidas en la segunda etapa
Leer	Lee un dato desde un dispositivo de entrada, por lo general del teclado.		Los identificadores a leer deben estar tabulados como

			entradas en la segunda etapa.
Asignar	Asigna un valor o resultado de una expresión de cálculo o lógica a un identificador.		Todos los identificadores debieron ser previamente inicializados.
Condición	Evalúa una expresión lógica para determinar uno de dos posibles flujos en la secuencia del algoritmo.		Todos los identificadores debieron ser previamente inicializados.
Repetir Hasta	Repite un bloque de elementos del diagrama exactamente n veces.		El identificador utilizado en los tres elementos que conforman esta estructura debe ser el mismo.
Hacer Mientras	Repite un bloque de elementos del diagrama de cero a n veces mientras la expresión lógica evaluada sea verdadera.		Cualquier identificador utilizado debe estar previamente inicializado.
Hacer Hasta	Repite un bloque de elementos del diagrama de una a n veces mientras la condición sea falsa.		Cualquier identificador utilizado debe estar previamente inicializado.

Es importante resaltar que la mayoría del software existente para editar diagramas de flujo permiten que el usuario pase por alto las reglas de construcción de dichos diagramas. Por otro lado, el software analizado en la sección de trabajos relacionados garantizan una construcción correcta, sin embargo no permiten la edición de los tres tipos de estructuras cíclicas, solo de una de ellas. El prototipo propuesto debe garantizar la correcta construcción de estos tres tipos de estructuras cíclicas, consideramos que esto es importante para evitar confusiones conceptuales a la hora de llevar lo visto en el salón de clase al software de apoyo.

Finalmente, la última etapa referente a probar el algoritmo debe permitir la ejecución del diagrama de flujo paso a paso para revisar que la solución cumple de forma correcta con lo que solicitó de inicio el enunciado del problema (ver Figura 6(d)). Al ejecutar el algoritmo en términos del diagrama de flujo editado en la etapa previa, el software

debería generar una corrida de escritorio similar a la que el profesor realiza en el salón de clase, mostrando en todo momento el valor actual de las variables así como el historial de los valores que tomaron previamente, de tal suerte que pueda rastrearse el porqué del valor actual. En resumen, esta etapa debe validar que a partir de los datos iniciales se obtienen los resultados esperados.

CONCLUSIONES Y TRABAJO FUTURO

Este trabajo presenta un *método para la enseñanza de algoritmos centrado en dos dimensiones*. A través de esta propuesta buscamos cubrir dos aspectos que consideramos relevantes en la enseñanza de los algoritmos; la resolución de problemas y la capacidad de abstracción.

De esta manera, aquellos estudiantes ajenos al concepto de algoritmos, podrá ir desarrollando su capacidad de resolución de problemas de manera incremental. Se busca que se avance gradualmente desde un nivel de conocimientos conceptual hasta que el estudiante sea capaz de evaluar requerimientos completos de un sistema. Buscamos evitar que el estudiante se enfrente en una primera instancia a solucionar problemas fuera de su comprensión, usando un paradigma de programación y el aprendizaje forzoso de un lenguaje de desarrollo para su expresión; que es lo que comúnmente se hace en cursos y libros de introducción a la programación.

El prototipo de software descrito aquí permite dar una idea sobre la relevancia que pueden tomar los recursos de apoyo en el proceso de enseñanza aprendizaje, más aún cuando estos recursos se desarrollan ad-hoc en base a una necesidad educativa concreta, apegados a una metodología, y formando parte de un esquema integral de aprendizaje.

Consideramos que contar con los tres elementos que propone nuestro método, permitirá al estudiante mejorar su desempeño y sus probabilidades de éxito en cursos avanzados dentro del área de programación.

Sin embargo, para garantizar la eficacia del mismo, será necesario llevar a cabo pruebas cuantitativas que evidencien en ambientes reales, el uso y resultado de lo aquí descrito, lo cual es parte de un trabajo futuro.

BIBLIOGRAFÍA

[1] *The Computing curricula 2005: The overview report*. Technical report, ACM, AIS, IEEE-CS, 2005.
[2] D. P. Ausebel. *Psicología educativa. Un punto de vista cognoscitivo*. 2a. Ed. Ed. Trillas. México, 2000.
[3] Duane Buck and David J. Stucki. *Design early considered harmful: Graduated exposure to complexity and structure based on levels of cognitive development*. SIGCSE Bull., 32(1):75-79, 2000.
[4] Ricardo A. Baeza-Yates. *Teaching algorithms*. SIGACT News, 26(4):51-59,1995.
[5] Eleni Voyiatzaki, Christos Christakoudis, Meletis Margaritis, and Nikolaos Avouris. *Teaching algorithms in secondary education: A collaborative approach*. In Proceedings ED Media 2004, AACE Publ, pages 2781-2789, 2004.
[6] John C. Giordano and Martin Carlisle. *Toward a more efective visualization tool to teach novice programmers*. In SIGITE '06: Proceedings of the 7th conference on Information technology education, pages 115-122, New York, NY, USA, 2006. ACM.
[7] James H. Cross II, T. Dean Hendrix, Jhilmil Jain, and Larry A. Barowski. *Dynamic object viewers for data structures*. In SIGCSE, pages 4-8. ACM, 2007.
[8] Duane Buck and David J. Stucki. *Jkarelrobot: a case study in supporting levels of cognitive development in the computer science curriculum*. SIGCSE Bull., 33(1):16-20, 2001.

[9] Osku Kannusmäki, Andrés Moreno, Niko Myller, and Erkki Sutinen. *What a novice wants: Students using program visualization in distance programming course*. In Ari Korhonen, editor, Proceedings of the Third Program Visualization Workshop (PVW 2004), Research Report CS-RR-407, pages 126-133, Warwick, UK, 2004. Department of Computer Science, University of Warwick.
[10] James H. Cross II, Larry A. Barowski, T. Dean Hendrix, and Joseph C. Teate. *Control structure diagrams for ada 95*. In TRI-Ada, pages 143-147, 1996.
[11] Stephen H. Edwards. *Using software testing to move students from trialand-error to reflection-in-action*. In SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education, pages 26-30, New York, NY, USA, 2004. ACM.
[12] Anany Levitin. *A new road map of algorithm design techniques*. Dr. Dobb's review, 25(4):23-24, 26-28, April 2000.
[13] Joyce R. Bruce. *Modelos de enseñanza*. Ed. Gedisa, 2002.
[14] Carlos Iván Chesñevar, Ana Gabriela Maguitman, María Paula González, and María Laura Cobo. *Teaching fundamentals of computing theory: A constructivist approach*. Journal of Computer Science and Technology, special issue on Computer Science Education (submitted), 2004.
[15] Seymour Papert. *Redefining childhood: The computer presence as an experiment in developmental psychology*. In IFIP Congress, pages 993-998,1980.
[16] David H. Jonassen, Chad Carr, and Hsiu-Ping Yue. *Computers as mind-tools*. TechTrends, 43(2):24-32, 1998.
[17] Alvaro H. Galvis. *Ambientes de eseñanza-aprendizaje enriquecidos con computador*. Revista de Informática Educativa, 1(2):117-145, 1988.
[18] D. Squires and A. McDougall. *Cómo elegir y utilizar software educativo*. Morata S. L., 2001.
[19] G. Polya. *How to Solve It*. Princeton Science Library Edition, 2004.
[20] Benjamin S. Bloom and David R. Krathwohl. *Taxonomy of educational objectives, Handbook 1: Cognitive domain*. Addison Wesley Publishing Company, 1956.
[21] Oliver, R. *Exploring strategies for on-line teaching and learning*. Distance Education, 20(2), 240-254, 1999.

CURRICULUM VITAE

Omar Santiago Nieva García

Profesor-Investigador asociado "B" adscrito a la carrera de ingeniería en computación en la Universidad del Istmo desde marzo de 2007. Obtuvo la licenciatura en informática por el Instituto Tecnológico de Oaxaca en 1999 y la maestría en ciencias de la computación por el Laboratorio Nacional de Informática Avanzada en 2006 en Xalapa Veracruz. Ha trabajado en el área de base de datos desde 1993 y laborado en el sector privado como especialista en sistemas de información y desarrollo de software desde 1998. Sus líneas de investigación son software educativo, bases de datos inductivas y almacenes de datos.

J. Jesús Arellano Pimentel

Profesor-Investigador asociado "B" adscrito a la carrera de ingeniería en computación en la Universidad del Istmo desde julio de 2005. Obtuvo el grado de Ingeniero en Sistemas Computacionales por del Instituto Tecnológico de Morelia en 2001, y el grado de Maestría en Ciencias en Ingeniería Eléctrica opción Sistemas Computacionales en 2005 por la Universidad Michoacana de San Nicolás de Hidalgo. Sus áreas investigación e interés profesional son desarrollo de software educativo, compiladores, robótica móvil, reconstrucción 3D y sistemas operativos.