Experiencia en el Desarrollo y Uso de un Software para Enseñar Algoritmos

J. Jesús Arellano[1], Omar Nieva[1], Gestemaní Arista[1], Rocío Solar[2]

[1]Ingeniería en Computación, Universidad del Istmo, Sto. Domingo Tehuantepec, Oaxaca
jjap@sandunga.unistmo.edu.mx, omarng@bianni.unistmo.edu.mx, arloget@hotmail.com
[2]División de Estudios de Posgrado, Universidad del Istmo, Sto. Domingo Tehuantepec, Oaxaca
solgr@sandunga.unistmo.edu.mx

Resumen. El presente trabajo detalla el desarrollo de un prototipo basado en la heurística de resolución de problemas de Polya desde una perspectiva de diseño MVC. Validaciones hechas frente a grupo permiten observar que el prototipo supera inconvenientes presentados en un escenario tradicional de enseñanza-aprendizaje de algoritmos a nivel universitario. Incluso supera otros inconvenientes de herramientas de software similares ampliamente utilizadas. Además, a partir de las validaciones del prototipo se han identificado nuevas características que permitirán mejorarlo notablemente en una nueva versión.

Palabras clave. Enseñanza de Algoritmos, Heurística de Polya, Diseño MVC, Prototipo de Software.

1 Introducción

La ACM [1] propone como parte esencial de los planes y programas de estudio de carreras con perfil computacional el estudio de los algoritmos. En dicho documento se hace evidente que los estudiantes relacionados con el área de las ciencias computacionales, e incluso ingenierías, necesitan aprender a programar, y en consecuencia obtener conocimientos básicos de algoritmos.

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. En [2] se propone una metodología para lograr este objetivo. Como parte de esta metodología se concibe un software de apoyo didáctico, dicho software es presentado en [3] desde una perspectiva centrada más en el usuario que en su desarrollo. La necesidad de desarrollar este tipo de software surge debido a que la mayoría de las herramientas existentes asumen que el estudiante sabe como analizar un problema y esbozar una solución en instrucciones algorítmicas.

El uso de software en el proceso de enseñanza-aprendizaje tiene dos alternativas a seguir, por un lado es posible elegir uno existente, por otro lado se puede construir uno a la medida. La elección del software no es una tarea sencilla, en [4] se estudia con mayor profundidad este aspecto. El problema de elegir software existente estriba

M.A. Cruz-Chávez (Ed): CICos 2011, ISBN. 978-607-00-5091-6. pp. 265 – 276, 2011

en encontrar uno que se ajuste lo mejor posible con los contenidos curriculares de lo que se pretenden enseñar. Por otro lado, el desarrollo de software a la medida tampoco es una tarea trivial, se deben considerar varios aspectos tales como: la necesidad educativa o curricular, componentes más apropiados en la interfaz de usuario, diseño, implementación, validación frente a grupo y retroalimentación, entre otros.

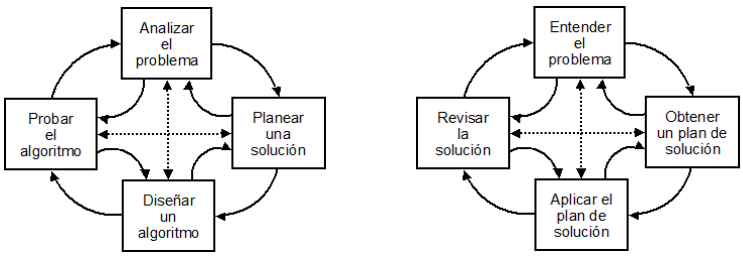
En este artículo se expone de forma breve y concisa la experiencia en el desarrollo y uso frente a grupo de un software hecho a la medida para enseñar algoritmos en cursos introductorios de programación a nivel superior. Así, en los siguientes apartados se describe el desarrollo del prototipo, el uso y validación del software frente a grupo, los resultados obtenidos, las características de una nueva versión del software y finalmente las conclusiones y trabajo futuro.

2 Desarrollo del prototipo

2.1 Contexto y definición de requisitos

Cuando los profesores desarrollan el software de apoyo en el proceso de enseñanza-aprendizaje de sus asignaturas, existe la ventaja adicional de ser su propio cliente. En este sentido los requisitos son más claros y se tiene una idea más precisa de cómo satisfacerlos.

En la metodología expuesta en [2] el principal objetivo es lograr un aprendizaje significativo en los estudiantes. Para este fin se propone un esquema de aprendizaje donde el apoyo didáctico fundamental es una herramienta de software basada en al heurística de resolución de problemas de Polya [5]. Dicha heurística consta de cuatro fases que son adaptadas de forma particular para estructurar el desarrollo y uso del software (ver Fig. 1), esta adaptación se resume la Tabla 1.



a) Fases de la Heurística de Polya b) Fases del Prototipo de Software

Fig. 1. Adaptación de la heurística de Polya a la estructura del Prototipo de software.

De la heurística adaptada se desprenden los principales requisitos del software. Clasificados según la fase de la heurística a la que pertenecen algunos de estos se enlistan a continuación:

1. Requisitos de la fase de analizar el problema:
   (a) Edición de los enunciados inicial y final del problema.
   (b) Presentación de una serie de 8 preguntas y edición de sus respuestas

   (c) Verificación del enunciado inicial, final y las 8 respuestas
2. Requisitos de la fase de planear una solución:
   (a) Registro de los elementos de entrada, salida y auxiliares de la solución propuesta
   (b) Tipificación y asignación de identificadores
   (c) Validación de los elementos
3. Requisitos de la fase de diseñar un algoritmo:
   (a) Generar un diagrama de flujo por omisión con los elementos registrados en la fase previa
   (b) Edición interactiva del diagrama de flujo
   (c) Ajuste del tamaño de los elementos del diagrama respecto al texto que contienen
   (d) Reacomodo automático del diagrama ante anidamientos y cambios de tamaño
   (e) Validación léxica, sintáctica y semántica del diagrama
4. Requisitos de la fase de probar el algoritmo:
   (a) Numeración de los elementos del diagrama de flujo
   (b) Ejecución paso a paso de los elementos
   (c) Visualización de la traza completa en la ejecución de cada paso

Tabla 1. Resumen de la adaptación de las fases de la heurística al prototipo de software.

Fases de la heurística de Polya	Descripción	Fases del prototipo de software	Descripción
Entender el problema	Tiene el propósito de entender el problema a través de una serie de preguntas como ¿Cuál es la incógnita? ¿Cuáles son los datos disponibles?, entre otras. Se puede hacer uso de esquemas, anotaciones o dibujos que faciliten la compresión del problema.	Analizar el problema	En esta fase se edita y analiza el enunciado inicial del problema con base en serie de 8 preguntas que deben ser resueltas a fin de obtener una descripción más detallada y comprensible del problema inicial.
Obtener un plan de solución	El objetivo es idear una estrategia que permita llegar de los datos disponibles hasta la resolución de la incógnita.	Planear una solución	Utilizando las respuestas de la fase anterior se planea la solución registrando los elementos de entrada, salida y auxiliares con su respectivo tipo de dato e identificador válido.
Aplicar el plan de solución	Aquí se debe llevar a cabo la estrategia sugerida verificando cada uno de sus pasos.	Diseñar un algoritmo	En esta fase se genera un diagrama de flujo inicial con los elementos previamente

			registrados. Dicho diagrama se edita a fin de construir una solución algorítmica al problema. Esta fase debe garantizar que el diagrama es gráfica, léxica, sintáctica y semánticamente correcto.
Revisar la solución	Se debe analizar la solución obtenida, a fin de estar seguros que es correcta y satisfactoria, es decir si el resultado es el que se esperaba.	Probar el Algoritmo	Una vez validado el diagrama de flujo de la fase anterior, la prueba del algoritmo consistirá en una corrida de escritorio que muestre la ejecución elemento a elemento del diagrama.

2.2 Arquitectura MVC y estrategia de desarrollo

Con el objetivo de satisfacer los requerimientos propuestos se adoptó como directriz de desarrollo el patrón Modelo Vista Controlador (MVC) [6]. De esta forma separamos el diseño de datos, el diseño de la interfaz de usuario y el diseño funcional en cada una de las cuatro fases del prototipo de software.

La estrategia de desarrollo implementada se apoya en el modelo de prototipado [7] con un enfoque incremental, de tal modo que se fueron desarrollando una a una las fases del prototipo agregándoles nuevas funcionalidades en cada iteración. Así pues, primero se diseñaron e implementaron los datos, después la interfaz de usuario y por último la lógica de cada nueva funcionalidad. En los siguientes apartados se describe con mayor detalle los diseños resultantes.

2.3 Diseño de datos (Modelo)

La principal estructura de datos en la fase de analizar el problema es un arreglo de 8 preguntas y respuestas, donde cada respuesta tiene asociado un estado: resuelta y sin resolver. Tanto las respuestas como los enunciados inicial y final son cadenas dinámicas. Por otro lado, las 8 preguntas son cadenas de tamaño fijo. En la fase de planear una solución la principal estructura de datos que se genera es una lista dinámica de variables y atributos. Cada nodo de la lista tiene la siguiente información: descripción de la variable, un tipo de variable (entrada, salida o auxiliar), un tipo de dato (entero, real o carácter), el nombre de la variable (identificador), un tipo de identificador (variable o constante), y valor de la variable.

Respecto a la fase de diseñar un algoritmo, la principal estructura de datos que se genera es un grafo dinámico para modelar un diagrama de flujo. Cada nodo del grafo almacena un tipo de elemento, código asociado, y dimensiones. Además, un nodo también almacena una serie de apuntadores que permiten organizar el grafo para modelar elementos secuenciales, estructuras de control y anidamientos. Finalmente en la fase de probar el algoritmo la principal estructura que se genera es una lista dinámica

de sentencias ejecutadas. Aquí cada nodo almacena un conjunto de cadenas, la primera contiene el número del elemento del diagrama ejecutado, las siguientes cadenas contienen los valores actuales de las n variables que se hayan utilizado en el algoritmo, y la última cadena contiene todo lo que el algoritmo escribe a pantalla.

2.4 Diseño de la interfaz de usuario (Vista)

El diseño de la interfaz de usuario permite interactuar con cada una de las fases de la heurística a través de cuatro pestañas, una para cada fase (ver Fig. 2).

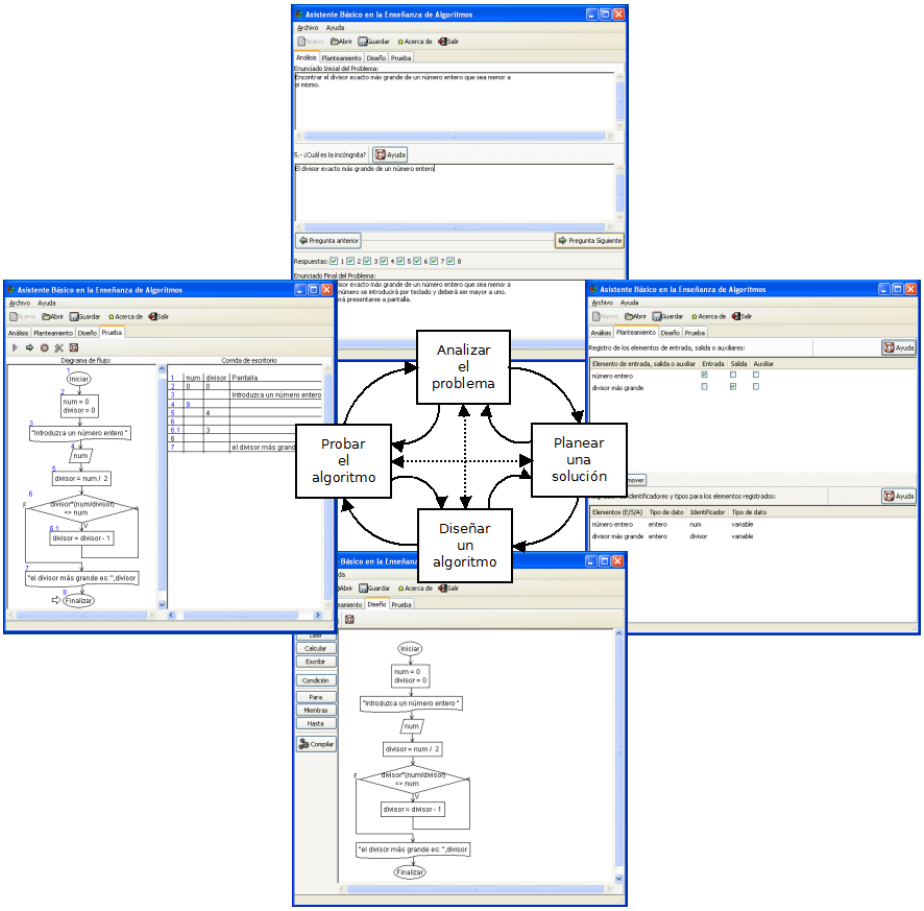


Fig. 2. Esquema general del diseño de la interfaz de usuario (Vista).

La pestaña de Análisis cuenta con cajas de edición para los enunciados inicial y final, así como para responder las 8 preguntas, también tiene dos botones para avanzar y retroceder entre las preguntas, además incluye 8 casillas de verificación que se activan cuando una respuesta es editada. En la pestaña de Planteamiento se tienen dos botones, uno para agregar variables (elementos) y otro para removerlas, por cada

variable adicionada se cuenta con tres casillas para indicar el rol de la variable en la solución algorítmica (entrada, salida, auxiliar), además esta pestaña incluye un apartado más para seleccionar el tipo de dato (entero, real o carácter), editar el identificador correspondiente y elegir el tipo de dicho identificador (variable o constante). En la pestaña de Diseño se cuenta con una serie de botones para insertar los distintos elementos que pueden conformar un diagrama de flujo y un área de dibujo para trazarlo, además se cuenta con un botón para compilar el código asociado a cada elemento de dicho diagrama. La pestaña de Prueba presenta del lado izquierdo el diagrama de flujo numerado y del lado derecho la ejecución de cada uno de sus elementos mostrando el valor actual de las variables en cada instante. Dicha ejecución se activa mediante la pulsación repetitiva del botón ejecutar paso a paso.

2.5 Diseño funcional (Controlador)

Funcionalidad de la fase Analizar el problema. Esta funcionalidad es bastante sencilla, el uso de los widgets tipo caja de edición va a permitir que el usuario edite los enunciados y las respuestas a las preguntas. Cada que se edite una pregunta y se avance para editar la siguiente, la lógica de control habilita la casilla de verificación de la pregunta recién editada, presenta la siguiente pregunta y limpia el widget para la edición de la nueva pregunta. En el momento que el usuario elija otra pestaña (fase), la lógica de control realiza una copia de todo lo editado en las estructuras de datos relacionadas con esta fase y valida que tanto los enunciados inicial y final, así como cada una de las 8 respuestas contengan texto, de no cumplirse esta condición se emite un mensaje de advertencia indicando que la fase no se concluyó satisfactoriamente.

Funcionalidad de la fase Planear una solución. Esta se dividió en 2 funciones específicas, resumiendo: registro de variables con sus atributos y validación de lista de variables. Al pulsar el botón agregar, la lógica de control genera un nuevo nodo con información por omisión y lo inserta en la lista de variables y atributos. Una vez que se inserta un nodo (variable) la interfaz visualiza su información en dos tablas. La primera tabla permite editar la descripción de la variable y elegir a través de casillas de verificación el rol de la variable. La segunda tabla permite elegir el tipo de dato de la variable, editar un identificador y elegir el tipo de identificador. También es posible remover un elemento, para ello habrá que seleccionarlo y pulsar el botón Remover, la lógica de control elimina el nodo correspondiente de la lista y actualiza la interfaz de usuario. Cuando un usuario elige la pestaña de Diseño, la lógica de control valida la lista de variables y atributos verificando que cada elemento cuenta con un rol, un tipo de dato y un identificador no repetido. Si la validación encuentra uno o más errores los notifica a través de mensajes de error.

Funcionalidad de la fase Diseñar un algoritmo. Esta también se dividió en 2 funciones específicas, una que engloba todo lo relacionado a la edición del diagrama de flujo y otra que se encarga de la validación del mismo. La lógica de control inicia generando un diagrama de flujo por omisión que puede contener dos o tres elementos. Tendrá dos elementos si la lista de variables y atributos esta vacía, tendrá tres elementos si no lo está. Dentro del diagrama de flujo se pueden insertar 7 tipos de elementos: Escribir, Leer, Calcular, Condición, Para, Mientras y Hasta. Los tres primeros elementos obedecen a la misma lógica de control por tratarse de elementos secuenciales,

de tal forma que su inserción genera un grafo de dos nodos que se adhiere al grafo principal, de forma similar ocurre con la inserción de otros elementos (ver Fig. 3).

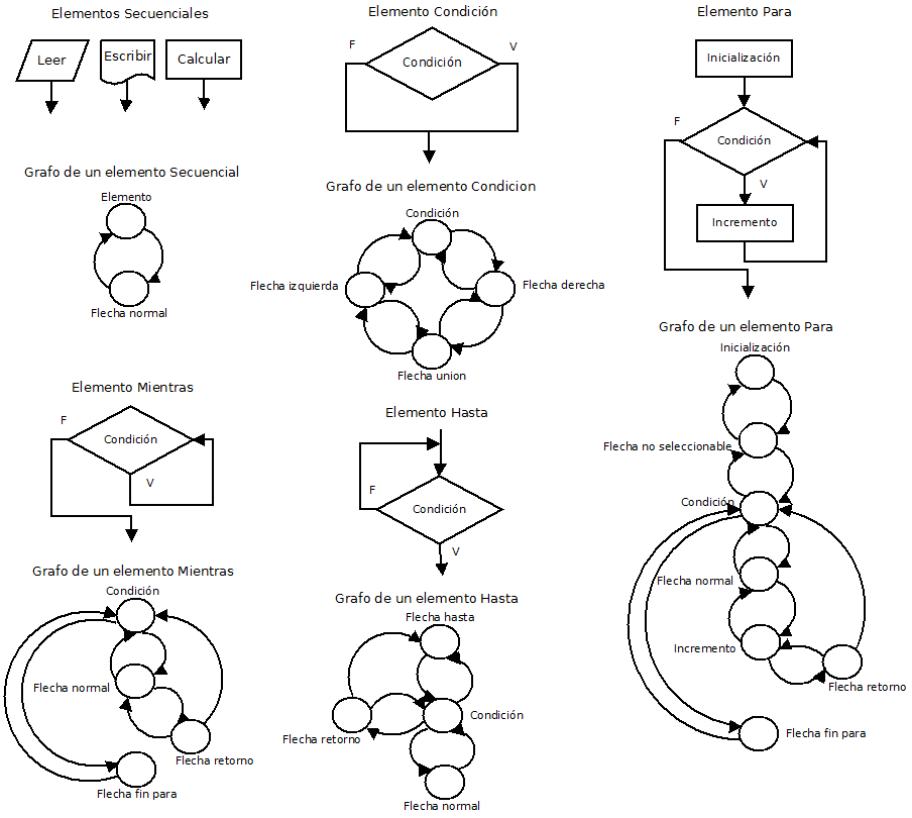


Fig. 3. Elementos del diagrama de flujo con sus correspondientes grafos.

Contar con varios nodos por elemento en el grafo facilita la lógica de control a la hora de recorrer el diagrama de flujo ya sea para construirlo, reacomodarlo, trazarlo o ejecutarlo. Ante cualquier inserción o modificación en el diagrama la lógica de control recalcula el tamaño y posición de los nodos de cada elemento de tal forma que al trazar el diagrama en el área de dibujo nunca superpongan sus elementos.

La segunda funcionalidad en esta fase inicia pulsando el botón compilar y consiste en validar el diagrama de flujo de forma léxica, sintáctica y semántica. La lógica de control consiste en un analizador léxico y un conjunto de gramáticas libres de contexto (GLC) de tipo LL1[8] para cada elementos del diagrama. Una ventaja particular de las gramáticas LL1 es que se puede utilizar la técnica de análisis por descenso recursivo [9], sin necesidad de emplear un generador de analizadores sintácticos o un autómata de pila. Además, la lógica de control para la validación semántica emplea la lista de variables y atributos como tabla de símbolos para verificar que las variables se utilicen correctamente según su rol y tipo de dato. En caso de encontrar un error léxico, sintáctico o semántico se notifica al usuario del error correspondiente a través de

un mensaje y mostrando de color rojo el código del elemento que produce el error. Si el diagrama no contiene errores se notifica al usuario que la compilación fue exitosa y podrá iniciar la siguiente fase para probar el algoritmo.

Funcionalidad de la fase Probar el algoritmo. Esta se dividió en 3 funciones específicas: numerar los elementos del diagrama, generar la lista de sentencias y visualizar la lista de sentencias. Aquí la lógica de control verifica que la fase previa este libre de errores. Siendo un diagrama de flujo completamente libre de errores, el primer paso para probarlo consiste en numerar sus elementos de forma que el número del elemento identifique su nivel de anidamiento. La numeración de los elementos del diagrama de flujo servirá para seguir la traza del algoritmo elemento a elemento. Al pulsar el botón de ejecución paso a paso se da inicio a la lógica de control que ejecuta los elementos del diagrama. Esta ejecución se realiza siguiendo un esquema de traducción dirigida por la sintaxis al estilo de un intérprete [8], el resultado final de dicha traducción es la información que va a contener el nuevo nodo de la lista de ejecución. Cada elemento del diagrama tiene su propia ejecución-traducción, en particular la ejecución de un elemento de tipo Leer requerirá la interacción con el usuario a través de un cuadro de dialogo que solicita el valor de una variable de entrada, la lógica de control del cuadro de diálogo verifica que el tipo de dato introducido corresponda con el tipo de dato de la variable. Siempre que un nuevo nodo es insertado a la lista, la lógica de control recalcula las posiciones donde han de trazarse los valores de las variables así como lo que debe presentarse en la columna que representa la Pantalla, es decir, ajusta la tabla de la traza del algoritmo para que nunca se superpongan los valores de las variables ni lo mostrado a pantalla.

2.6 Plataforma de implementación

Siguiendo la idea de generar un prototipo de software funcional conforme a la filosofía de Software Libre [10] se tomó la decisión de desarrollar el prototipo inicial para Linux Ubuntu [11] utilizando el lenguaje de programación C. Se emplearon las librarías de Gtk+, Gdk y Pango, además del diseñador de interfaces gráficas Glade. Las librerías y el diseñador de interfaces forman parte del proyecto Gnome [12]. Otro factor que influyó en la elección de la plataforma fue la compatibilidad de Gtk+ con la mayoría de las versiones de Windows a través del proyecto Glade/Gtk+ para Windows. Glade conjuntamente con la librería de Gtk+ provee los widgets (ventanas, botones, menús, etc.) suficientes para dotar a la interfaz gráfica de usuario con los elementos requeridos por el software. Por su parte, las librerías Gdk y Pango contienen las primitivas gráficas para dibujar y manipular texto respectivamente, estas se emplearon en la visualización del diagrama de flujo y la corrida de escritorio.

3 Uso y validación de software frente a grupo

Tradicionalmente en un curso de algoritmos ocurre el siguiente escenario: a) el docente plantea un problema representado por un enunciado textual o verbal, b) el alumno trata de encontrar la solución en términos de un diagrama de flujo o pseudocódigo, c) el alumno verifica con el profesor si su algoritmo es correcto. Desde nuestro punto de vista este escenario tradicional tiene principalmente dos inconvenientes: 1) el docente

no valora el análisis previo al diseño de la solución ya que solo ve el pseudocódigo o el diagrama de flujo, y 2) el alumno obtiene retroalimentación hasta que el profesor puede atenderlo para verificar su solución, lo que se convierte en un verdadero problema con grupos numerosos.

3.1 Uso de otras herramientas de software disponibles

En una primera instancia, se utilizó software existente con la finalidad atenuar el inconveniente 2 referente a la retroalimentación del alumno. Particularmente se empleo el software Raptor [13] y DFD [14] cada uno en un grupo de estudiantes diferente durante el Ciclo Escolar 2009-2010B en un curso propedéutico de algoritmos de nivel universitario. En ambos casos se encontró que el software no da soporte alguno para las fases de análisis y planteamiento del problema, por lo que se mantiene el inconveniente 1 antes mencionado. Por otro lado, si bien el uso de estas dos herramientas contribuyó a subsanar el problema de la retroalimentación, encontramos otros dos inconvenientes en su utilización:

- Raptor soporta solo un tipo de estructura cíclica (Hasta) y DFD soporta solo dos (Para y Mientras), esto impide que se ejerciten las tres estructuras cíclicas de la programación estructurada ocasionando errores conceptuales a la hora de determinar cuál es el tipo de ciclo más apropiado en la solución algorítmica
- Los diagramas de flujo editados en Raptor y DFD no respetan a cabalidad la notación estándar de los diagramas de flujo para representar estructuras de control, esto genera confusión en los alumnos a la hora de implementar los diagramas de flujo de sus libros de consulta. Además un diagrama hecho en DFD no se ajustan en función del contenido de sus elementos, lo que impide visualizar la lógica completa del algoritmo.

3.2 Resultados del prototipo

El prototipo se utilizó en un curso propedéutico de algoritmos a nivel universitario en el Ciclo Escolar 2010-2011B. Los esquemas de diseño empleados en el desarrollo del software lograron superar los inconvenientes mencionados anteriormente. En el transcurso del propedéutico se atendieron dos grupos, el A y el B con 14 y 13 alumnos respectivamente. El grupo A empleo el prototipo, el grupo B no empleo software de apoyo. Al respecto se observo lo siguiente (Fig. 4):

- El grupo A mostró mayor interés en resolver los problemas de tarea cuando empezaron a utilizar el software, de hecho el 93% del grupo entregó todas sus tareas. Por otro lado, solo el 46% de los alumnos del grupo B entregaron sus tareas.
- El 100% de alumnos del grupo A manifiesta que el uso de la herramienta los ayudó de forma significativa en el análisis y diseño de las soluciones algorítmicas a los problemas planteados. Por otro lado, a pesar de no haber utilizado una herramienta de software durante su curso, en el grupo B el 84% opina que usar una herramienta los ayudaría de forma significativa a en el análisis y diseño de soluciones algorít-

micas, el 8% opina que le ayudaría poco, y el %8 restante no tiene idea de si le puede ayudar o no.

- Al finalizar el propedéutico el 57% del grupo A aprobó el curso de algoritmos. Por otro lado, solo el 38% del grupo B aprobó el curso. En parte estos resultados se deben a que la materia se evaluó considerando 80% examen y 20% tareas.

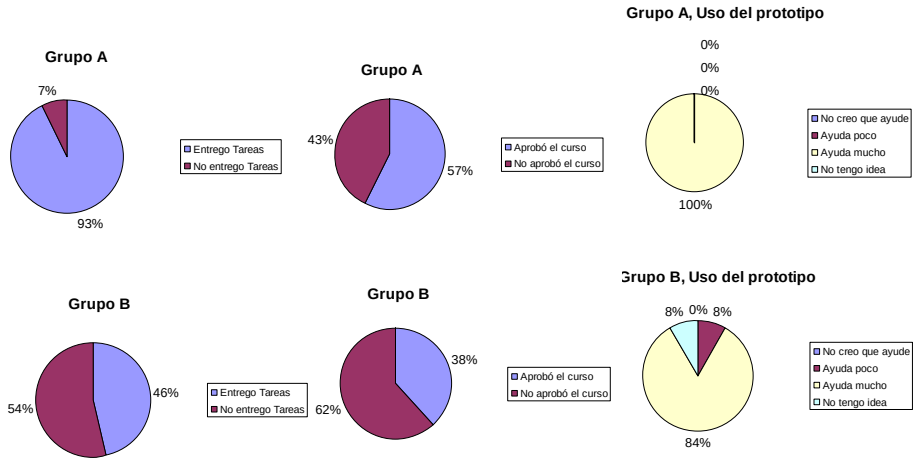


Fig. 4. Gráficas comparativas de los Grupos A y B (el grupo A utilizó el prototipo).

4 Nueva versión del software

No obstante que los resultados preliminares fueron positivos, también se detectó la necesidad de mejorar el prototipo inicial con un conjunto de nuevas características. Estas son:

- Soporte para variables de tipo arreglo y soporte para variables de tipo lógico.
- Traducción del diseño formal del algoritmo dado en diagrama de flujo a pseudocódigo y código fuente en C.
- Diferenciar mediante color las áreas validas de inserción en el diagrama.
- Manejo de nombres largos en el manejo de los archivos generados por el software.
- Soporte para la inicialización de variables y constantes.
- Mejorar la interacción con los mensajes de error.
- Visualizar mediante el uso de distintos colores el anidamiento de estructuras de control en un diagrama de flujo.
- Empleo de funciones básicas de librería en las operaciones de cálculo. Funciones tales como raíz cuadrada, seno, coseno, etc.
- Mayor portabilidad entre distintos sistemas operativos.

Actualmente se esta desarrollando la nueva versión del prototipo utilizando el Proceso Unificado [15], el lenguaje de programación Java con la plataforma NetBeans [16] y

técnicas de ingeniería inversa [7] sobre el prototipo inicial afín de obtener una nueva versión mejorada del software.

Conclusiones y trabajo futuro

En este trabajo presentamos el desarrollo y uso de un software empleado en la enseñanza de algoritmos a nivel universitario. El diseño MVC de dicho software permitió construir un prototipo, que por un lado satisface los requerimientos derivados de la adaptación de la heurística de resolución de problemas de Polya, y por otro, supera los inconvenientes de herramientas de software similares respecto al soporte en las fases de análisis y planeación de una solución algorítmica. Además el modelo del diagrama de flujo a través de un grafo proporciona la capacidad de representar, trazar y ejecutar todas las estructuras de control básicas en cualquier curso de algoritmos.

El empleo del prototipo frente a grupo demuestra lo importante que es contar con apoyos de aprendizaje que permitan a los estudiantes ejercitar los conocimientos adquiridos en el aula. Sin embargo, es importante aclarar que el uso de software por si solo no basta, este debe ser parte de un esquema de aprendizaje que integre, además del software, contenidos y actividades de aprendizaje. Actualmente se cuenta con los contenidos (recursos de aprendizaje). En un trabajo futuro se planean construir actividades de aprendizaje que exploten todas las capacidades del nuevo prototipo de software.

Referencias

[1] The Computing curricula 2005: The overview report. Technical report, ACM, AIS, IEEE-CS (2005)

[2] Nieva, G. O., Arellano, P. J.: Método de enseñanza centrado en 2 dimensiones. En: 4º Simposio Internacional de Sistemas Telemáticos y Organizaciones Inteligentes, pp. 881-897. Editorial FESI. Xalapa Veracruz, México (2009)

[3] Arellano, P. J., Nieva, G. O.: ABEA, Herramientas de apoyo en la Enseñanza de Algoritmos y habilidades de programación. En: VI Semana Nacional de Ingeniería Electrónica, p.p. 193-202. Huajuapan de León, Oaxaca México (2010)

[4] Squires, D., McDougall, A.: Cómo elegir y utilizar software educativo. Editorial Morata S. L. (2001)

[5] Polya, G.: How to solve it. Princenton Science Library Edition (2004)

[6] Trygve Reenskaug: THING-MODEL-VIEW-EDITOR - an Example from a planningsystem. Notas Técnicas, Xerox PARC (1979)

[7] Pressman, R. S.: Ingeniería del software. Un enfoque práctico. Editorial Mc Graw Hill. Quinta edición (2002)

[8] Aho, A. V., Sethi, R., Ullman, J. D.: Compiladores. Principios técnicas y herramientas. Editorial Pearson (1990)

[9] Louden K. C.: Construcción de compiladores. Principios y práctica. Thomson (2005)

[10] GNU Operating System. La definición de Software Libre, http://www.gnu.org/philosophy/free-sw.es.html

[11] Ubuntu. Canonical Ltd, http://www.ubuntu.com/
[12] The GNOME Project, http://www.gnome.org/
[13] Carlisle, M. C., Wilson, T. A., Humphries, J. W., Hadfield, S. M.: RAPTOR: introducing programming to non-majors with flowcharts. Journal Computing Small Coll (2004)
[14] Cárdenas, V. F., Castillo I. N., Daza C. E.: Editor e intérprete de algoritmos representados en diagramas de flujo. Red Iberoamericana de Informática Educativa nodo Colombia (1998)
[15] Stephen R. S. Análisis y diseño orientado a objetos con UML y el Proceso Unificado. Editorial Mc Graw Hill (2005)
[16] NetBeans. Oracle. http://netbeans.org/