Revista Digital Universitaria
1 de diciembre 2011 • Volumen 12 Número 12 • ISSN: 1067-6079

ARTICULO

VISUALIZACIÓN DE PROGRAMAS UTILIZANDO TÉCNICAS ORIENTADAS A ASPECTOS

I.SC. Juan Alberto Hernández Martínez
y Dr. Ulises Juárez Martíneza

© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

1 de diciembre 2011 • Volumen 12 Número 12 • ISSN: 1067-60710

Visualización de programas utilizando técnicas orientadas a aspectos

Resumen

La Visualización de Programas es un área de la computación que ayuda ampliamente a la comprensión de algoritmos, mostrando resultados intermedios de toda la ejecución del mismo. En este sentido, contar con herramientas que proporcionen este tipo de funcionalidad, es muy útil dentro del área educacional. Lograr la obtención de dichos resultados implica el indagar en lugares estratégicos dentro del algoritmo para recuperar los datos pertinentes, siendo las variables locales el punto de importancia. Es necesario tener el soporte que permita adentrarnos en la búsqueda de información dentro de un programa dado y así obtener la visualización adecuada que permita mejorar la comprensión del mismo. El problema esencial en la Visualización de Programas es la falta de soporte para indagar adecuadamente en las estructuras de control y en las variables locales, lo que ocasiona que las soluciones se implementen de forma local a un programa, dificultando aún más la comprensión. La Programación Orientada a Aspectos ofrece una alternativa para este tipo de situaciones, identificando, separando y encapsulando asuntos. Por su parte el framework Énfasis da un soporte orientado a aspectos para aplicar cortes sobre variables locales, lo que en conjunto permiten obtener visualizaciones más robustas y entendibles.

Palabras clave:

visualización de programas, algoritmos, variables locales, programación orientada a aspectos, énfasis.

Program Visualization UsingAspect-OrientedTechniques

Abstract

Program visualization is a computing area that aids to understand algorithms showing intermediate results about the execution of itself, in this sense, having tools that provide this type of functionality is very useful in an educational area. To achieve such results it is necessary to search in strategic places inside the code to get the relevant information. Being local variables the locus, it is necessary to have the support that allows recover important data inside a program, and so, to get a correct visualization that allows a better understanding about an algorithm. The main trouble in Program Visualization is the lack of support to inquire appropriately on control structures and local variables, causing that solutions are implemented locally for each program getting hard the understanding even more. Aspect-Oriented Programing offers an alternative to this kind of situations, identifying, separating and encapsulating concerns. Énfasis framework gives an aspect-oriented support to apply pointcuts on local variables, which

© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

allow getting both better visualizations and better comprehension.

Keywords:
program visualization, algorithms, local variables, aspect-oriented programming, énfasis.

Introducción

La programación siempre ha requerido de un nivel de análisis demasiado alto, sin importar el paradigma sobre el cual se trabaja, siendo la estructura de datos una de las áreas que más complejidad demandan. Es muy difícil en la mayoría de los casos saber con exactitud lo que realiza cada algoritmo. Considerado este un problema que afecta en gran medida al aprendizaje de alumnos, debemos considerar que es importante el darle una óptima solución. La Visualización de Programas aborda en gran parte este tipo de situaciones, tratando de hacer visible el funcionamiento de un programa dado, sin embargo, la forma en como se desarrollan las herramientas que pueden dar una solución, no es la mejor.

En este artículo se destacan algunos puntos importantes en el enfoque de la Visualización de Programas, como una buena opción para mejorar el entendimiento de algoritmos. Se presentan algunas herramientas centradas en esta área computacional y se hace mención al paradigma de la Programación Orientada a Aspectos, como alternativa para el desarrollo de aplicaciones enfocadas a la Visualización de Programas, sin alterar el código fuente de los programas originales.

Desarrollo

Porqué la Visualización de Programas

Si bien ya se ha mencionado anteriormente que la programación implica un alto nivel de abstracción, sin importar el paradigma sobre el cual se trabaja, una buena alternativa para lograr esa comprensión del funcionamiento de algoritmos es la Visualización de Programas. En demasiadas ocasiones, el usuario dedicado a la programación se enfrenta a pequeños, medianos y grandes códigos, los cuales cuentan con un grado de complejidad bastante considerable. En algunas ocasiones se puede claramente saber cuál es toda la serie de pasos que se siguen para que el resultado devuelto por el mismo sea el correcto, pero ¿qué pasa cuando el usuario necesita saber el funcionamiento de algoritmos que implican recursividad o estructura de datos? Lo más probable es que sea demasiado complejo saber a ciencia cierta qué hace el programa cada vez que se ejecuta y, por consecuencia, no se tenga la total seguridad de entenderlo. Dadas estas situaciones, la Visualización de Programas funge como área auxiliar a dichas problemáticas, sirviendo en muchas ocasiones como un aspecto importante en términos educacionales, dado que permite al usuario entender de manera clara

© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

1 de diciembre 2011 • Volumen 12 Número 12 • ISSN: 1067-60710

programas complejos.

Visualización de Programas

La Visualización de Programas presenta vistas del código fuente del programa y de las estructuras de datos. La visualización estática del programa realiza un análisis del texto del mismo y provee la información que es válida para todas las ejecuciones independientemente de su entrada. Los sistemas de Visualización de Programas emplean editores sintácticos u optimizadores de código que producen un mejoramiento de la impresión, como indentado, diferencias de colores entre palabras reservadas y otros identificadores, la representación gráfica de la estructura del programa y de los datos; pero no pueden explicar la conducta del programa, ya que esto depende de los datos de entrada además del código mismo. Una representación dinámica del programa provee información acerca de la ejecución de un programa específico, sobre un conjunto de datos de entrada. Se puede realizar, por ejemplo, destacando las líneas de código fuente cuando éstas están siendo ejecutadas, estableciendo la pila de invocaciones de procesos y mostrando cómo los valores de los datos cambian mientras el programa se está ejecutando (Moroni y Señas, 2002).

Herramientas para Visualización de Programas

Pérez y Velázquez (2009) reportan la aplicación SRec, la cual aporta varias vistas complementarias sobre la ejecución de programas recursivos. La aplicación cuenta con múltiples opciones y facilidades educativas, que convierten su manejo en una actividad sencilla e intuitiva, minimizando su tiempo de implantación y aprendizaje, y facilitando así la labor docente de manera notable.

Almeida, et al. (2009) presentan a EDApplets, una aplicación Web orientada a la enseñanza/aprendizaje de la programación y la algorítmica en las ingenierías. Basada en la tecnología de Applets Java, está orientada a la animación y la visualización mediante trazas de algoritmos y estructuras de datos. La herramienta permite cubrir diversos aspectos en una enseñanza que está dirigida a distintos tipos de estilos de aprendizaje: activo/reflexivo, metódico/intuitivo, visual/oral, etcétera.

Soler y Lezcano (2009) presentan el sistema de visualización de programas VisualProg, que consta de tres funcionalidades principales: edición, ejecución y animación de programas escritos en Sucel, un lenguaje sencillo con similitud al estilo de programación C/C++/Java. Wannda et al. (2001) presentan un ambiente de programación gráfico interactivo para Windows, el cual es llamado Alice. Ofrece una completa secuencia de comandos y ambiente de prototipos para el comportamiento de objetos 3D. Soporta recursividad como medio para crear

© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

animaciones más poderosas, con acciones repetidas que acercan al usuario cada vez más a una tarea completa.

Diferencia entre depuración y visualización

Cuando se habla de Visualización de Programas, inmediatamente se asocia con los depuradores, causando en la mayoría de las ocasiones mucha confusión y, sobre todo, una mala definición acerca del área mencionada. Este tipo de confusiones viene dado principalmente por la cercanía que existe entre la Visualización de Programas y la Depuración, sin embargo ambas áreas tienen un objetivo bastante diferente. El principal objetivo de la Depuración es eliminar los errores de los programas. En general, la Depuración es un proceso que busca detectar, diagnosticar y corregir los errores en un programa dado. La depuración permite mostrar el valor que toma en determinado momento una variable dentro de un código especifico, sin embargo, no está planteado para enseñar al usuario y dar la posibilidad de que obtenga un conocimiento general del funcionamiento de un algoritmo. Simplemente muestra información en puntos específicos, para posteriormente corregirlos en caso de ser necesario.

Ejemplificando lo anterior, podríamos suponer un código que permita ordenar un arreglo de números, en términos de Visualización de Programas. Visualizar al usuario toda una traza de resultados intermedios en cada iteración durante la ejecución del programa, es una de las alternativas. De esta manera el usuario puede ver cómo y cuál fue el cambio de valor en cada iteración para las variables involucradas dentro del código, desde el inicio de la ejecución hasta su final. Si la ordenación es un éxito o no, eso es algo que la visualización no puede solucionar, dado que su principal objetivo es mostrar el funcionamiento de un algoritmo en ejecución. Retomando el mismo ejemplo en términos de Depuración, la situación es diferente. Una vez que sabemos que nuestro programa ordena mal los números, se indica cierto punto dentro de nuestro código para ver si ahí posiblemente se hacen mal las cosas, pero en ningún momento nos hemos percatado de cuál es el funcionamiento del código. Simplemente estamos intentando reparar algo que está mal. He aquí una notable diferencia entre ambas áreas, y es por este detalle que un depurador no es considerado una herramienta que visualice programas.

Programación Orientada a Aspectos (POA)

La Programación Orientada a Aspectos (POA) es un paradigma que posibilita la separación de asuntos y proporciona una clara vía para encapsular funcionalidad entrecruzada, la cual hace a los programas más difíciles de leer, mantener, entender y reutilizar. La lógica detrás de la POA es que los sistemas informáticos son programados mejor por separado, especificando e implementando los variados aspectos del sistema y una descripción de sus relaciones. Mecanismos en el ambiente subyacente de la POA, son responsables de tejer o componer los

© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

1 de diciembre 2011 • Volumen 12 Número 12 • ISSN: 1067-60710

diferentes aspectos en un programa coherente.
Los aspectos pueden extenderse desde conceptos de alto nivel, como seguridad y calidad del servicio hasta conceptos de bajo nivel como autenticación, entre otros. Pueden ser funcionales, como características o reglas de negocio; o no funcionales, como sincronización y manejo de transacciones (Elhaibe, 2006).

Entre las ventajas que presenta dicho paradigma se encuentran las siguientes (Reina, 2000):

- Un código menos enmarañado, más natural y más reducido;
- Una mayor facilidad para razonar sobre los conceptos, ya que están separados y tienen una dependencia mínima;
- Más facilidad para depurar y hacer modificaciones en el código;
- Se consigue que un conjunto grande de modificaciones en la definición de un concepto tenga un impacto mínimo en las otras, y
- Se tiene un código más reutilizable para acoplarlo y desacoplarlo cuando sea necesario.

Aportación de la POA

Para visualizar un programa se necesita: un código fuente a visualizar y un código que permita mostrar la visualización del algoritmo al usuario. Retomando el ejemplo del ordenamiento de un arreglo de números, imaginemos que necesitamos visualizar el funcionamiento de nuestro código en una ventana de SWING. Ya tenemos nuestro código fuente que se encarga de ordenar un arreglo de números y el cual queremos visualizar, y ya contamos con un código que tanto muestra la ventana de SWING como realiza las operaciones para poder visualizar la información en la misma. Este último realiza dos operaciones: crear la ventana y visualizar el funcionamiento del programa. Dicho de esta manera, claramente se aprecia que el código se encuentra mezclado. Si necesitáramos cambiar algunas sentencias, resultaría difícil el mantenimiento del mismo. Si se quisiera agregar alguna nueva funcionalidad, probablemente se tenga que cambiar gran parte del mismo, dado que solamente fue realizado para un código en especial, por lo que reutilizarlo sería muy complicado. Problemas de este tipo son los que tienen solución gracias a las ventajas que la POA proporciona, pues nos permite separar todas esas funcionalidades en códigos totalmente ajenos unos con otros, dejando más claro y entendible el programa, con mayor reutilización y mayor capacidad de mantenimiento, entre otras.

Visualización del método BubbleSort

Para ilustrar el ejemplo mencionado con anterioridad presentamos lo siguiente:

- Se tiene una clase, en este caso, llamada Bubble, la cual contiene un método bubbleSort(int[] arreglo) que recibe como parámetro un arreglo de enteros y cuya función es

© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

ordenarlo.

```
01    public class Bubble {
02        public static int[] bubbleSort(int[] arreglo){
03            intalmacen, i=0, j=0;
04            for(i = 0; i <arreglo.length; i++){
05                for(j = 0; j < i; j++){
06                    if(arreglo[i] <arreglo[j]){
07                        almacen = arreglo[j];
08                        arreglo[j] = arreglo[i];
09                        arreglo[i] = almacen;
10                    }
11                }
12            }
13            return arreglo;
14        }
15    }
```

- Nuestra intención era visualizar toda la ejecución del programa, desde su inicio hasta su fin en una ventana de SWING, para lo cual presentamos el código de la clase VentanaDatosDos. Aquí simplemente se muestra el método inicializarComponentes(), el cual se encarga de crear una ventana que en su parte superior contiene una etiqueta que por el momento no dice nada, y en la parte inferior un botón con la leyenda Salir.

```
public void inicializarComponentes(){
        panelPrincipal = new JPanel();
        panelPrincipal.setLayout(new BoxLayout(panelPrincipal,BoxLayout.Y_AXIS));
        panelPrincipal.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

        lblMensaje = new JLabel();
        lblMensaje.setFont(tipoLetra);
        lblMensaje.setForeground(Color.black);

        btnBoton = new JButton("Salir");
        btnBoton.setFont(tipoLetra);
        btnBoton.addActionListener (new Cerrar());

        panelPrincipal.add(lblMensaje, BorderLayout.NORTH);
```


© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

1 de diciembre 2011 • Volumen 12 Número 12 • ISSN: 1067-60710

```
        for(int i=0; i<vector.size();i++)
                panelPrincipal.add((JTextField)vector.elementAt(i));

        panelPrincipal.add(btnBoton, BorderLayout.SOUTH);
        this.getContentPane().add(panelPrincipal);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("Traza del arreglo");
        this.setSize(400,400);
        this.setLocation(200,200);
        this.setResizable(true);
        this.setVisible(true);
}
```

- Dentro de esta misma clase VentanaDatosDos, sobresale otro fragmento de código que en conjunto con el anterior forman toda la clase. Este segundo fragmento simplemente contiene sus método getTraza() y setTraza(String traza) para recibir la información que se visualizará en la ventana, y un método llamado creaCaja() cuya función es crear un JTextField por cada línea de información recibida.

```
public void setTraza(String traza){
        this.traza = traza;
}

public String getTraza(){
        returnthis.traza;
}

public void creaCaja(){
        JTextFieldtxtDatos = new JTextField();
        txtDatos.setForeground(Color.black);
        txtDatos.setFont(tipoLetra);
        txtDatos.setText(this.getTraza());
        vector.addElement(txtDatos);
}
```

- Lo siguiente que tenemos es una clase llamada Principal que simplemente realiza la

© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

ejecución del programa. Dentro del método main solamente se envía el arreglo de números al método que se encarga de ordenarlo, en nuestro caso el bubbleSort.

```
public class Principal {
        public static void main(String[]args){
                intarreglo[] = {3,7,2,5,9,4,12,1};
                int v[] = Bubble.bubbleSort(arreglo);
        }
}
```

Hasta este momento cada clase tiene su propia funcionalidad. La primera es ordenar un arreglo de números; la segunda es crear una ventana en SWING, y la tercera es ejecutar el programa. La parte interesante está dada a continuación, para lo cual tenemos lo siguiente:

- Se ha realizado una clase llamada BubbleAspectMediator, que se encargará de realizar la visualización. Lo importante es que las técnicas utilizadas dentro de ella están orientadas a aspectos. Este pequeño código localiza puntos estratégicos de la clase Bubble para que cuando se ejecute el método bubbleSort(int[] arreglo), inmediatamente recupere la información que nos permita ver cómo van cambiando los valores del arreglo en cada iteración de inicio a fin, y de esta manera enviarla a la clase que se encargará de crear la ventana en SWING.

```
public class BubbleAspectMediator extends Aspect{
        Pointcut corteTraza2 = new Lines("Bubble","{6,13};");
        String mensajeTraza2 = "VentanaDatosDos.capturaTraza(java.util.Arrays.
toString(arreglo));";
        Advice avisoTraza2 = new After(mensajeTraza2);
        Pointcut corteEjecucion2 = new Execution("int[] Bubble.bubbleSort(int[])");
        String bodyEjecucion2 = "VentanaDatosDos.visualizaPantalla();";
        Advice avisoEjecucion2 = new After(bodyEjecucion2);

        Intertype campo2 = new Field("VentanaDatosDos", "public static String cadena=\"\";");
        String nuevoMetodo = "public static void capturaTraza(String contenido){" +
"v.setTraza(contenido);" + "v.creaCaja();" + "}";
        Intertype metodoCapturaTraza2 = new Method("VentanaDatosDos", nuevoMetodo);
        String nuevoMetodo2 = "public static void visualizaPantalla(){" +
"v.inicializarComponentes();" + "}";
```


© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

1 de diciembre 2011 • Volumen 12 Número 12 • ISSN: 1067-60710

```
        Intertype metodoVisualiza2 = new Method("VentanaDatosDos",nuevoMetodo2);

        public void weaving(){                                  intertyping(campo2,metodoC
apturaTraza2,metodoVisualiza2);
                advising(avisoTraza2,corteTraza2);
                advising(avisoEjecucion2,corteEjecucion2);
        }
}
```

Este código tiene algunos elementos importantes que es necesario destacar para su entendimiento total:

1. El código anterior está realizado mediante un framework llamado Énfasis, el cual está diseñado especialmente para programar extensiones orientadas a aspectos de baja granularidad. Mediante éste se pueden realizar cortes sobre variables locales y cortes sobre la estructura estática de las clases para agregar miembros e instrumentación de clases (Juárez, 2008).
2. Los elementos Pointcut indican las líneas en las cuales actuará el aspecto. Para este código se indicaron tanto Lines("Bubble","{6,13};"), y Execution("int[] Bubble. bubbleSort(int[])"), como los puntos sobre los cuales deberá actuar el mencionado.
3. El elemento Intertype permite introducir nuevos campos a las clases, los cuales no fueron contemplados desde el inicio.
4. El método weaving() es el lugar donde se controla el aspecto con la clase correspondiente, para la cual fue diseñado.

El resultado es el siguiente:

© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

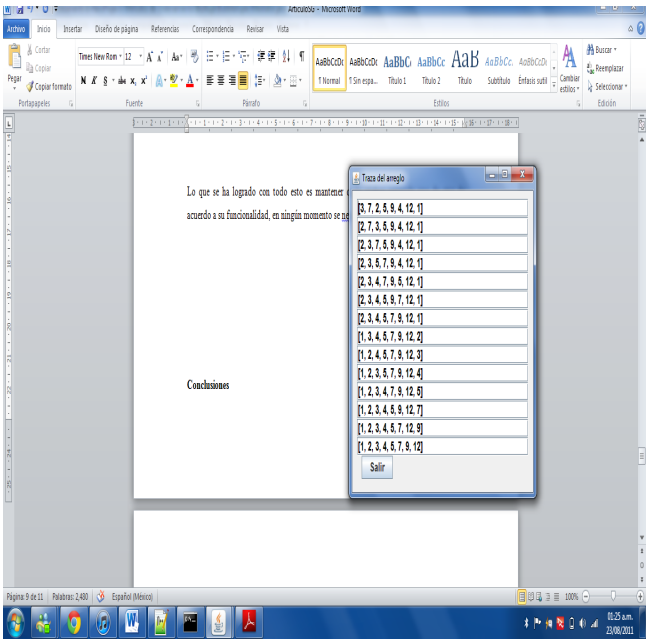

FIGURA 1. Visualización de la traza de ejecución del método BubbleSort

Lo que se ha logrado con todo esto es mantener cada código separado uno del otro, de acuerdo a su funcionalidad. de esta manera obtenemos una gran limpieza entre ellos, un mejor entendimiento, mayor capacidad de reutilización y así evitamos los problemas típicos a la hora de desarrollar aplicaciones.

Conclusiones

La Visualización de Programas es una muy buena opción para satisfacer las necesidades de entendimiento y comprensión de algoritmos, disminuyendo considerablemente el nivel de análisis y haciéndolos mucho más fáciles de entender, lo cual resulta muy difícil de realizar a simple vista, debido a la complejidad que actualmente tiene la programación, sin importar el paradigma sobre el cual se trabaja. De esta forma, resulta demasiado benéfico el poder hacer uso de dicha área computacional. Aunado a ello, utilizar el paradigma de la Programación Orientada a Aspectos es sin lugar a dudas una alternativa atractiva para realizar aplicaciones, sin tener los típicos problemas con los cuales se cuenta en algunos otros paradigmas, como lo son código mezclado, código disperso y duplicidad de código, por mencionar algunos. Tanto la Visualización de Programas

© Coordinación de Acervos Digitales. Dirección General de Cómputo y de
Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.

1 de diciembre 2011 • Volumen 12 Número 12 • ISSN: 1067-60710

como la Programación Orientada a Aspectos, mantienen un punto en común: darle al usuario la posibilidad de tener una mejor comprensión de lo que realiza. Mientras que la primera en mención lo hace por medios visuales, la segunda lo logra con la limpieza que se tiene al mantener las funcionalidades en códigos separados.

Referencias

Almeida, Francisco, et al. “EDApplets: Una Herramienta Web para la Enseñanza de Estructuras de Datos y Técnicas Algorítmicas,” 2009, universidad de La Laguna Tenerife.

Dann, Wanda, et al. “Using Visualization to Teach Novices Recursion,” 6th Annual Conference on Innovation and Technology in Computer Science Education, 2001, new York,USA.

Elhaibe, Rodrigo. “Metodologías Ágiles y Programación Orientada a Aspectos, Estudio de una posible integración,” Noviembre 2006, Universidad de Morón, Buenos Aires Argentina.

Juárez, Ulises, “Énfasis: Programación Orientada a Aspectos de Grano Fino,” Ph.D.dissertation, CINVESTAV, Octubre 2008, México D.F.

Moroni, Norma y Señas, Perla. “SVED: Sistema de Visualización de Algoritmos,” 2002, laboratorio De Investigación y Desarrollo en Informática y Educación (LIDInE) Buenos Aires.

Pérez, Antonio y Velázquez, Ángel. “Animación Automatizada de Técnicas de Diseño de Algoritmos,” XV Jornadas de Enseñanza Universitaria de la Informática (JENUI), Julio 2009, Barcelona.

Reina, Antonia. “Visión General de la Programación Orientada a Aspectos,” Diciembre 2000, facultad de Informática y Estadística Universidad de Sevilla.

Soler, Yolanda y Lezcano, Mateo. “Ambiente de ayuda al diseño de programas de computadoras y determinación de su eficiencia,” Revista de Informática Educativa y Medios Audiovisuales, 2009, vol. 6(13).

© Coordinación de Acervos Digitales. Dirección General de Cómputo y de Tecnologías de Información y Comunicación -UNAM
Se autoriza la reproducción total o parcial de este artículo, siempre y cuando se cite la fuente completa y su dirección electrónica.