La Visualización de Algoritmos como Recurso para la Enseñanza de la Programación

Norma Moroni - Perla Señas
Grupo LIDInE
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur.
Av. Alem 1253 - 8000 - Bahía Blanca - ARGENTINA
moroni@criba.edu.ar - psenas@csuns.edu.ar

Introducción.

En la búsqueda de nuevas estrategias metodológicas para la enseñanza de la programación, es interesante el paralelo que se puede realizar con los estudios sobre el aprendizaje de la lecto-escritura del lenguaje natural. Es muy común que al enseñar programación se centre la atención en la creación de algoritmos, descuidando el trabajo de lectura y comprensión de algoritmos existentes. Trabajar de esa manera puede pensarse en cierta forma equivalente a tratar de enseñarle a un alumno a escribir dejando de lado los contenidos de lectura mecánica y sobre todo de lectura comprensiva [27] [28].

En relación con la enseñanza de la programación se destaca la importancia de las actividades comprendidas en:

- construcción de algoritmos
- lectura comprensiva de algoritmos

Existe consenso en considerar un enfoque que parta de la resolución de problemas, para el primer ítem, y que se apoye en técnicas de visualización de algoritmos y programas para el segundo caso. Se aborda en este trabajo el estudio realizado sobre visualización de software, especialmente pensado para la lectura comprensiva de algoritmos y programas.

La visualización de software es el uso de gráfica de computadoras y animación interactiva para ilustrar y presentar programas, procesos y algoritmos. Se basa en el uso de diseño gráfico, de animación, de sonido, de video y tiene como característica sobresaliente la interacción entre el usuario y la computadora.

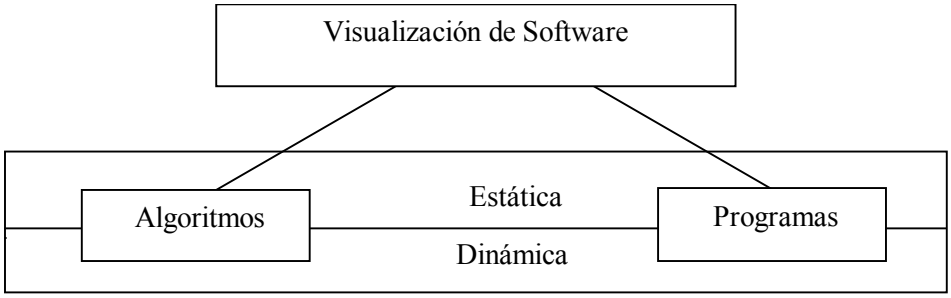
La visualización de la conducta dinámica de los programas y de sus modelos abstractos, los algoritmos, y consecuentemente, el beneficio psicopedagógico de su aplicación en la enseñanza, constituyen una gran área de investigación que aún no ha sido completamente explorada.

La motivación especial que condujo a realizar este trabajo es la necesidad de mejorar las herramientas existentes para facilitar la comprensión de los aspectos de la conducta dinámica de los algoritmos en las distintas fases del ciclo del software, en el diseño, en el desarrollo, en la búsqueda de errores, en la depuración y en el mantenimiento.

Generalmente, para lograr una comprensión acabada de un programa se realiza, por un lado, el estudio del código fuente, lo que es incómodo y de dificil aplicación y, por otro lado, la construcción de casos de prueba para explicar la conducta de un programa, lo que es una tarea penosa y especulativa. Estas dificultades motivan el desarrollo de programas especiales que son usados para ayudar a explicar la conducta de otros programas [1].

Visualización de software

La Visualización de Software comprende la visualización interactiva de dos áreas fundamentales: la de algoritmos y la de programas, que en ambos casos, se pueden considerar transversalmente como presentaciones estáticas o dinámicas.



La visualización de algoritmos consiste en la exposición de abstracciones de alto nivel que describen semánticamente el software. La representación estática de los mismos está basada en la esquematización de la estructura de su especificación. La representación dinámica, que habitualmente se llama animación de algoritmos, muestra su comportamiento en tiempo de corrida. La visualización estática de programa realiza un análisis del texto del mismo y provee la información que es válida para todas las ejecuciones independientemente de su entrada [15] [11]. Una representación dinámica del programa provee información acerca de su ejecución sobre un conjunto de datos de entrada [15]. La información de la ejecución se puede ofrecer en forma instantánea, o en un tiempo posterior (análisis post mortem). El análisis post mortem puede realizar computaciones grandes para condensar la información de la ejecución y presentarla en una forma útil. Los métodos no son excluyentes.

Las herramientas en tiempo de corrida pueden ser pasivas o interactivas. En un sistema pasivo, la herramienta presenta información al usuario, pero el usuario tiene poco control sobre la actividad del mismo; en un sistema interactivo, el usuario puede tener control externo sobre la información que se está exhibiendo, puede estar disponible para modificar la computación que está siendo observada o bien el dato que está siendo procesado [11].

Los sistemas para comprender programas son usados en una variedad de aplicaciones. El debugger es uno de los programas más comunes que ayudan a la comprensión del software. Otras de las aplicaciones de sistemas de comprensión de programas es la encargada de poner a punto la efectividad o performance. Una tercera aplicación de la comprensión es la instrucción y orientación del software, que resulta de interés para aprender importantes algoritmos, estructuras de datos o técnicas de programación. Con las herramientas de comprensión de programas las personas afectadas pueden beneficiarse por la información provista por ellas, antes de consultar el código fuente.

Antecedentes de la visualización de software.

La preocupación frente a la dificultad en la comprensión del software ha sido constante. Entre las distintas mejoras se pueden mencionar las orientadas hacia la estructura lógica y expresiva del estilo de los programas, tal como el diseño top down y el refinamiento paso a paso [14], la programación estructurada [15], la modularidad [16], y las herramientas de software [17]; en la claridad y potencia expresiva de los lenguajes de programación [18], y en el desarrollo de aproximaciones orientadas a objeto para diseño y desarrollo [19].

El embellecimiento de la escritura, el uso de espaciado, identación, y disposición para que el código fuente fuera más fácil de leer son algunas de las primeras ideas de visualización que surgieron en un principio. Los Prettyprinters son programas que sistemáticamente identan el código fuente de un programa de acuerdo a su estructura sintáctica. SEE Program Visualizer [3] es un sistema que automáticamente establece un programa en C de acuerdo a una guía de estilo basada sobre principios de diseño. El sistema WEB mejora la escritura de programas combinando el texto del programa fuente con la documentación en un sola publicación usando un lenguaje sofisticado [2].

A medida que se progresa en la capacidad de representación gráfica de las computadoras, se desarrolla software que permite visualizar el comportamiento de los programas. El primer trabajo

que ilustra la ejecución de un programa es el que presenta Baecker a mediados de los 70 [3]. El sistema Balsa es uno de los primeros que permite la creación de las visualizaciones de algoritmos genéricos en Pascal [4]. El sistema agrega al código primitivas de animación las cuales activan las funciones de representación gráfica que ofrece el sistema. A finales de los años 80 y en la década de los 90 surgen la gran mayoría de los sistemas que contemplan ahora la visualización de programas concurrentes o paralelos como los sistemas Voyeur [5] o Belvedere [6].
Como sucesores del sistema Balsa, surgen los sistemas Tango [7], Zeus [8] y [9]. Estos dos últimos permiten la visualización de ejecuciones de programas concurrentes. El sistema Pavane [10] permite la visualización de programas declarativos.

Visualización de algoritmos.

La Visualización de Algoritmos presenta un aspecto distintivo de los otros tipos de visualizaciones ya que no existen entidades tangibles a las cuales asirse para diseñar la visualización. Se debe decidir lo que se quiere visualizar y cómo se va a obtener la información necesaria para hacerlo [23].
En las visualizaciones de algoritmos, en las que se utilizan grandes niveles de abstracción, puede resultar difícil especificar la apariencia visual de los objetos lógicos presentes. En este caso el sistema de visualización debe ofrecer un conjunto de transformaciones y entidades gráficas para realizar la especificación.
La visualización del algoritmo en cualquiera de sus dos formas, dinámica o estática, puede ser muy difícil de relacionar con las construcciones del programa que la codifican y que constituyen las especificaciones de su comportamiento. Es decir, se presenta allí una brecha entre la representación del algoritmo y la especificación del programa subyacente [4].

Contribución pedagógica de la Visualización de Algoritmos

Se debe disponer de sistemas amigables con el usuario (facilidad de interacción hombre-máquina, en cuyo diseño se debe poner especial interés en la percepción humana). Pero, además, la experiencia de los usuarios con el lenguaje de programación y su nivel de familiaridad con la propia visualización de software resultante, influyen sobre la calidad del sistema de visualización.
La Visualización de Algoritmos presenta un medio audio-visual interactivo multimedial cuyas bondades en el campo de la psicopedagogía han sido ampliamente comprobadas. La Visualización de Algoritmos es una disciplina joven y, por ello, tanto la experiencia en sí como la evaluación de los resultados deben ajustarse a ella. Las animaciones de algoritmos presentan importantes beneficios educativos [24] [25] [26]:

- Logran un incremento de la motivación.
- Facilitan el desarrollo de destrezas.
- Asisten en el desarrollo de habilidades analíticas.
- Ofrecen un buen soporte al docente.
- Permiten la exploración, de las peculiaridades de un algoritmo, jugando interactivamente.

Distintos centros educativos emplean estos sistemas como apoyo al aprendizaje [13] [24].

Sistema de Visualización de Algoritmos propuesto

Se propone un Sistema de Visualización Interactiva de Algoritmos orientado a una programación estructurada y modular.
El sistema debe proveer multiventanas en las que se debe exhibir la estructura estática del algoritmo, la estructura estática de los datos, el menú de selección de la representación de datos y aquellas ventanas que permiten el ingreso de los valores de los datos y de su representación con los cuales se realizará la corrida del programa.

Durante la ejecución del programa, el sistema debe permitir la visualización del dinamismo de la activación de los distintos módulos en ejecución, la historia del paso por los distintos módulos y fundamentalmente, el comportamiento del algoritmo a través de la animación de los datos. Ventanas adicionales, deben estar activas para permitir la interacción del usuario con la visualización de manera que pueda interrumpirla, fijar la imagen, modificar su velocidad de presentación, modificar los valores de entrada, modificar la representación de los datos y sus colores, activar la audición ya sea de errores o de pasos claves. La audición ofrece algunas ventajas respecto de la observación permanente de las ventanas activas. Escuchar un sonido determinado ayuda al usuario a detectar una acción cuya ejecución se está llevando a cabo en ese momento, sin necesidad de observar la pantalla.

Conclusión

La animación de algoritmos y la visualización de programas ayudan a los estudiantes a comprender los conceptos de software y también a los docentes en su tarea de enseñar dichos conceptos.
Independientemente de la calidad de un sistema de visualización se deben tener en cuenta algunos factores importantes que influyen sobre la efectividad del mismo, como son la experiencia de los usuarios con el lenguaje de programación y su nivel de familiaridad con la propia visualización de software resultante.
Actualmente se están estableciendo los lineamientos básicos de un sistema de visualización de algoritmos basados en un ambiente estructurado y modular, que constituya un entorno de aprendizaje efectivo para ayudar a los estudiantes en la comprensión y desarrollo de algoritmos y programas.

Bibliografía

[1] Jeffery, Clinton L. *Program Monitoring and Visualization*. Springer-Verlag. 1999.
[2] Knuth, D. *Computer-drawn Flowcharts*. Communication of the ACM. 1963.
[3] Baecker, R. *Two systems Which Produce Animated Representatios of the Execution of Computer Programs*.1975. ACM SIGCSE Bulletin.
[4] Brown y Sedgewick. *A system for Algorithm Animation*. ACM Computer Graphics (Proc of SIGGRAPH'84) 1984-1985.
[5] Socha, D. Bailey, M. Y Notkin, D. *Voyeur: Graphical Views of Parallel Programs*. Conf. On Parallel Processing, 1987.
[6] Hough,A. y Cuny, J. *Initial Experiences with a Pattern-Oriented Parallel Debugger*. Int. Conf. On Parallel Processing. 1987.
[7]Stasko, J**.** *Tango : A framework and system for Algorithm Animation*. IEEE Computer. 1990.
[8] Brown, M. *Zeus: A System for Algorithm Animation and Multi-view Editing***.** Technical report SRC-75, Digital- Systems Research Center. 1992]
[9] Stasko, J y Kraemer, E. *A Methodology for Building Applicatios Specific Visualizations of Parallel Programs*. Journal of Parallel and Distributed Computing 1992.
[10] Roman , G. Cox. K, Wilcox, C, y Plun J. *Pavane: a System for Declarative Visualization of Concurrent Computations*. Journal of Visual Languages and Computing. 1992.
[12] Stasko, J., Domingue, J., Brown, M., Price, B. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, 1998.
[13] Bazik, j. Tamassia, R., Reiss, S y van Dam, A. *Software Visualization in Teaching at Brown University.* 1996.
[14] Wirth N. *Program Development by Stepwise Refinement*. Communitations of the ACM. 1971.
[15] Conroy K. y Smith R., *NEATER2: A AP/I Source Statement Reformatter*. Communitations of the ACM. 1972.
[16] Parnas D. y Weiss, D**.** *Active Design Reviews: Principles and Practices*. Proceedings of the 8th International Conference on software Engineering. 1985.
[17] Kernighan B y Plauger P. *Software Tools*. Adisson-Wesley, Reading, MA, 1976.

[18] Wirth N. *Modula: A Language for Modular Multiprogramming. Software- Practice and experience*. 1977.
[19] Booch G. *Object Oriented Design with Applications*, Benjamin/Cummings,1984.
[20] Wasserman A. *Tutorial: Software Development Environments*. IEEE Computer Society Press, 1981.
[21] Dart, S.; Ellison, R.; Freiler, P y Habermann. *A Software Development Environments*, Computer 1987.
[22] Chifosky, E. y Rubenstein, B. *Categorisation and representation of phisics problems by experts and novices*. Cognitive Science, 1981.
[23] Miller, G. *What to Draw? When to Draw? An Eassy on Parallel Program Visualisation*. Journal of Parallel and Distributed Computing.1993.
[24] Lawrence, A., Brade, A., Stasko, J*., Empirically Evaluating the Use of Animations to Teach Algoritms*. Technical Report GIT-GVU-94-07, Graphics, Visualisation, and Ussability Center, College of Computing. Georgia Institute of Technology. 1994.
[25] Price, B.; Beacker, R; y Small, I. *An Introduction to Software Visualisation.* Software Visualisation. MIT Press. 1998.
[26] Merril, P. at all . *Computers in education*. Allyn & Bacon. 1996.
[27] Álvarez Méndez, J. *Didáctica de la lengua materna*. Madrid. Akal. 1987.
[28]Contreras, D. *Enseñanza, profesorado y curriculum*. Madrid. Akal. 1994.