
Universidad Rey Juan Carlos

# Generación Semiautomática de Animaciones de Programas Funcionales con Fines Educativos

**TESIS DOCTORAL**

**Jaime Urquiza Fuentes**

**2007**


**Universidad Rey Juan Carlos**

Escuela Superior de Ingeniería Informática

Departamento de Lenguajes y Sistemas Informáticos I

# Generación Semiautomática de Animaciones de Programas Funcionales con Fines Educativos

**Tesis Doctoral**

**Director:**

Dr. D. Jesús Ángel Velázquez Iturbide

**Doctorando:**

D. Jaime Urquiza Fuentes

2007

Dr. D. Jesús Ángel Velázquez Iturbide, Catedrático de Universidad del Departamento de Lenguajes y Sistemas Informáticos I de la Universidad Rey Juan Carlos director de la Tesis Doctoral "*Generación Semiautomática de Animaciones de Programas Funcionales con Fines Educativos*" realizada por el doctorando D. Jaime Urquiza Fuentes,

HACE CONSTAR

que esta Tesis Doctoral reúne los requisitos necesarios para su defensa y aprobación.

En Móstoles a 31 de octubre de 2007,

Dr. D. Jesús Ángel Velázquez Iturbide

# Agradecimientos

*En primer lugar querría agradecer a mi director de Tesis todo el esfuerzo que me ha dedicado y el apoyo que me ha prestado. Creo que ha sido un estupendo director de Tesis, cuya experiencia ha enriquecido tanto esta tesis, como mi investigación en general.*

*Este trabajo ha sido posible gracias a la colaboración prestada por otras personas, como Alberto con su PFC, Patxi con sus desarrollos, los alumnos de la universidad que participaron en las distintas evaluaciones, los miembros del departamento de Lenguajes y Sistemas Informáticos I, en especial a Carlos por su ayuda durante las sesiones de evaluación, y la gente del servicio de alumnos de la universidad. También quiero agradecer las interesantes dicusiones con la gente del EdTech, en especial a Andrés, Roman, y Niko; así como a Carolina, Javi, Jauma y el resto de la "Orda hispano americana" de Joensuu por su acogida durante aquellos meses.*

*Gracias a mi familia, por el apoyo incondicional y la paciencia que han tenido conmigo. Gracias a Ceci, por estar siempre ahí. Gracias a Laura, por recordarme las cosas que son realmente importantes. Y gracias a mis amigos, gracias por vuestra amistad, apoyo y comprensión.*

# Índice general

# Índice de figuras

# Índice de tablas

# Resumen

En esta tesis doctoral presentamos un modelo para la generación de animaciones de programas funcionales. Los puntos clave de este modelo son su simplicidad, el poco esfuerzo que los usuarios deben dedicar a la construcción de las animaciones y las ventajas de su uso educativo.

Las animaciones de programas son visualizaciones dinámicas de su comportamiento. Una de las ventajas más importantes que aportan las tecnologías de visualización es ayudar a la formación de modelos mentales, más aún si lo que se visualiza es un concepto abstracto. Los conceptos que los alumnos deben aprender sobre programación funcional son totalmente abstractos; por lo tanto, las animaciones de programas funcionales ayudarán a los estudiantes a comprender su funcionamiento, facilitando su aprendizaje.

El modelo de construcción propuesto se compone de un conjunto de tareas, unas automatizables y otras manuales, por ello es un proceso semiautomático. Su simplicidad y el poco esfuerzo necesario se consiguen automatizando al máximo el proceso, y haciendo que las tareas donde se necesita la intervención del usuario sean lo más sencillas posibles.

Las visualización de programas y algoritmos no es un campo nuevo, tiene más 35 años. Sin embargo, el uso educativo de las visualizaciones se ha basado durante bastante tiempo en la creencia, sin respaldo empírico, de su eficacia como herramientas pedagógicas. Actualmente, existen resultados que muestran beneficios pedagógicos en el uso de las animaciones en situaciones concretas. Desde un punto de vista general, coge fuerza la idea de que las animaciones ayudan en el aprendizaje si se utilizan de forma activa, no como meras herramientas de consulta. Por ello centramos nuestro interés en el uso educativo de la construcción de animaciones.

Los trabajos presentados utilizan un modelo ya existente de ejecución, visualización y animación de programas funcionales. Aunque proporcionan ventajas tecnológicas destacables, dichos modelos no han podido demostrar su aportación en el ámbito educativo.

Diseñando un proceso de construcción de animaciones simple y que requiera poco esfuerzo, permitiremos que los estudiantes centren su atención en las tareas educativas, p.ej., producción de animaciones representativas del comportamiento del programa. Esta es la base para hacer de la construcción de animaciones una herramienta que mejore el aprendizaje.

El proceso de construcción se basa en la selección de los distintos fotogramas que componen la animación. Cada fotograma es la visualización estática de un estado de la ejecución del programa. Como la animación representará la ejecución del programa, nos hemos centrado en ofrecer una visión global de dicha ejecución mostrando tantas visualizaciones como se pueda. Para ello reducimos las visualizaciones estáticas manteniendo su comprensión por parte del usuario. Las versiones reducidas son de gran calidad gracias a la inclusión de nuevas características en las visualizaciones y a un algoritmo de reducción de imágenes eficaz y eficiente. Por otro lado, proporcionaremos al usuario la posibilidad de tener más detalles sobre una visualización en concreto sin perder la información contextual de la ejecución del programa. Esto lo conseguimos diseñando una nueva técnica de visualización de información llamada *R-Zoom*, que permite al usuario ver detalles de visualizaciones manteniendo el contexto, de forma eficaz, eficiente y satisfactoria. Como resultado final, el proceso de construcción en sí requiere poco esfuerzo, con lo que permitiremos a los estudiantes centrar su atención en las tareas educativas.

Para situarnos en una posición realista, hemos mejorado las animaciones tanto en su utilización –ahora son recursos utilizables vía Web– como en sus contenidos –añadiendo explicaciones textuales

que complementarán las representaciones gráficas–. También hemos creado un modelo para la gestión de colecciones de estas animaciones. Para poder detectar realmente las aportaciones pedagógicas de este modelo de construcción, lo hemos comparado con una metodología típica sin uso de animaciones y otra que sólo las usa como material de consulta. Los resultados muestran que el uso de animaciones –consultándolas o construyéndolas– mejora los resultados académicos de los estudiantes, pero que la construcción de animaciones motiva mucho más a los estudiantes en relación con la asignatura donde se usan.

En conclusión, proponemos la construcción de animaciones de programas funcionales con nuestro modelo como tarea educativa, complementada con la disponibilidad de colecciones de animaciones para su consulta.

# Abstract

In this dissertation we present a generation model for functional program animations. The main ideas of this model are its simplicity, its effortlessness and the advantages of its educational use.

Program animations are dynamic visualizations of its behavior. One of the most important advantages of visualization technologies is that they assist to building mental models, specially if we visualize an abstract concept. The functional programming concepts that students have to learn are absolutely abstract; therefore, functional program animations will help students to understand these concepts, facilitating learning them.

The proposed construction process is made up of a number of tasks. Some of them can be performed automatically, but other are manual. This is why we speak about a semiautomatic process. Making the process as much automatic as possible, and designing manual tasks as simple as possible, help in reaching simplicity and effortlessness of the process.

Program and algorithm visualization is not a novel research field, it is more than 35 years old. However, during a long time its educational use has been based on the believe of its pedagogical effectiveness, without empirical evidence. Nowadays there exist results showing learning outcomes in some situations. From a general point of view, the idea that animations help to learn if they are used in a more active scenario, not only as viewing material, is becoming stronger. This is why we focus our attention in the educational use of animation construction tasks.

The works presented use an existing model for execution, visualization and animation of functional programs. Although they provide notable technologic advantages, these models have not demonstrated their effectiveness from an educational point of view.

If we design a simple and effortless animation construction process, we will allow students to focus on the educational tasks, e.g., production of representative program animations. This is the base for making animation construction an effective educational tool.

The construction process is founded on the selection of the snapshots that form an animation. Each snapshot is the static visualization of an execution stage of the program. As the animation has to represent the execution of the program, we have focused our attention on offering a global view showing as many visualizations as possible. We face this by reducing the static visualizations while keeping enough degree of comprehension. We produce high quality reduced versions by including new features in the visualizations and using an effective and efficient image reduction algorithm. In addition, we allow the user to see more details of each visualization keeping the context information of the execution. To do this, we have designed a novel information visualization technique called *R-Zoom*. It allows the user to see details of one visualization plus its context at the same time, in an effective, efficient and user satisfying way. The result is an effortless construction process that allows students to focus on the educational tasks.

To be down-to-earth we have improved both the use of animations –now they are Web resources– and their contents –integrating textual explanations–. Also, we have created a management model for collections of animations. To detect actual pedagogical advantages of this construction model, we have compared it with both: a typical learning scenario without the use of animations, and a just-viewing-animations scenario. Results show that using animations –viewing or constructing them– improve students academic results, but constructing improves students retention with the subject where animations are used.

Therefore, we propose animation construction based on our model as an educational task, augmented with the availability of collections of animations to view.

# Capítulo 1

# Introducción

*Esta tesis doctoral presenta un modelo semiautomático para la construcción de animaciones de programas funcionales con fines educativos. Las animaciones de programas son representaciones visuales –visualizaciones– dinámicas del comportamiento durante su ejecución. En el ámbito de esta tesis, los programas se codificarán y ejecutarán según el paradigma de programación funcional.*

*El proceso de construcción de estas animaciones se compone de un conjunto de tareas, unas automatizables y otras manuales, por ello es un proceso semiautomático. Automatizando al máximo el proceso, y haciendo que las tareas donde se necesita la intervención del usuario sean lo más sencillas posibles, conseguimos un proceso de construcción de animaciones que requiere poco esfuerzo por parte del usuario.*

*Las visualizaciones ayudan a la formación de modelos mentales de los conceptos visualizados. Por lo tanto, las animaciones de programas ayudarán a los estudiantes a comprender su funcionamiento, facilitando su aprendizaje. En esta tesis exploramos los beneficios educativos, tanto de las animaciones de programas funcionales como de su propio proceso de construcción.*

*En este primer capítulo describimos el marco global de la tesis, explicando las motivaciones, la hipótesis de trabajo, los objetivos que persigue y sus aportaciones principales. Finalmente hacemos un breve resumen de la estructura del resto de capítulos.*

## 1.1. Motivación

Podemos definir "visualizar" como *"dar representación visual a algo que no la tiene"*. Por ejemplo, se pueden visualizar las arterias que recorren el cerebro, que no vemos porque están ocultas dentro de la masa cerebral; o se puede visualizar el grado de humedad en el ambiente, que aunque sea un hecho de naturaleza física no es visible; o se puede visualizar la estructura organizativa de una empresa, que no vemos porque es un conjunto de entidades y relaciones abstractas. De forma intuitiva, podemos percibir que estas tres visualizaciones ofrecen ayudas en sus ámbitos correspondientes: médico, meteorológico y empresarial. De hecho, la principal ventaja de las visualizaciones es que ayudan a formar modelos mentales de los conceptos visualizados, mejorando su comprensión y posibilitando su aprovechamiento para la realización de tareas asociadas.

La *visualización del software* se asemeja a los dos últimos ejemplos que hemos mencionado. Por un lado tiene naturaleza física aunque no visible. El software es un conjunto de datos organizados en ficheros que, cuando se ejecutan, consiguen que un ordenador se comporte de cierta forma. Además,

el software en ejecución son un conjunto de señales eléctricas que recorren los diferentes componentes de un ordenador. Sin embargo para diseñar y producir software utilizamos un conjunto de conceptos abstractos como bucle, contador, bifurcación, recursividad, pila, array, etc. Ambas facetas son visualizables: la naturaleza física del software en ejecución se suele visualizar en los depuradores cuando nos muestran el estado del procesador, mientras que la naturaleza abstracta se puede visualizar en un diagrama de flujo. Ambas visualizaciones ayudan a los encargados de la producción del software en sus diferentes tareas.

Todo software trata de llevar a cabo una serie de tareas manejando ciertas estructuras de datos. La producción del software pasa por su codificación en un lenguaje de programación concreto. La *visualización de algoritmos* trata de dar representación visual al software en términos de las tareas y estructuras de datos, mientras que la *visualización de programas* lo hace en términos de las características concretas del lenguaje de programación utilizado.

La animación es la visualización dinámica, lo que significa que en las representaciones visuales la dimensión temporal también aporta algún significado concreto. La animación del comportamiento de programas y algoritmos (animaciones, de ahora en adelante) es un campo en el que se lleva trabajando más de 35 años [7] y ha tenido aplicación tanto en ámbitos profesionales como educativos. En el entorno profesional los problemas que se resuelven con las animaciones están bastante bien acotados y definidos. En el entorno educativo el objetivo principal está claro: las animaciones deben ayudar a los estudiantes en las tareas de aprendizaje; lo que no está claro es cómo conseguirlo.

Desde el principio, las animaciones se han usado en entornos educativos. Sirva como ejemplo la famosa película sobre algoritmos de ordenación *Sorting Out Sorting*[1] de R.M. Baecker [8, 9]. Las animaciones se han utilizado durante mucho tiempo asumiendo que suponían una ayuda para los estudiantes. Sin embargo, cuando los investigadores trataban de obtener evidencia empírica de sus ventajas como herramientas educativas, no encontraban resultados suficientemente representativos. Como dicen Stasko et al. [214]:

> "*Desafortunadamente, la viabilidad de las animaciones de algoritmos como herramientas de apoyo a la enseñanza permanece arraigada en la intuición. Nunca se ha presentado evidencia empírica sustantiva que pruebe estas ideas.*"

Así, las líneas de investigación pasaron a buscar qué características hacían que las animaciones fueran un recurso educativo eficaz. Se han enunciando propiedades como la abstracción de detalles que no aportan nada a los conceptos que se enseñan, la explicación de las representaciones gráficas usadas, o la posibilidad de retroceso durante la reproducción de la animación.

El trabajo de Hundhausen et al. [94] supuso un punto de inflexión en estas investigaciones, cambiando el objeto de interés de las animaciones en sí a la forma en que se usan:

> "*Nuestro descubrimiento más significativo es que la forma en que los estudiantes usan la tecnología de visualización de algoritmos tiene un mayor impacto en la eficacia que lo que les muestra la tecnología de visualización de algoritmos.*"

Las formas de uso de las animaciones por parte de los estudiantes se han clasificado de diferentes maneras [61, 158], pero siempre contemplando diferentes grados de implicación del estudiante con las animaciones. Algunos ejemplos son la respuesta a preguntas durante la reproducción de la animación

[1] http://www.kmdi.toronto.edu/rmb/video/sos_recap.mov

[78], la exploración del comportamiento de los algoritmos cambiando los datos de entrada [213], o la simulación del comportamiento de los algoritmos [118].

El paradigma de programación funcional también se ha visto afectado por los avances en la visualización del software, pero se han centrado en un ámbito más tecnológico. Los conceptos que un programador debe utilizar para construir programas funcionales son bastante abstractos. Por ejemplo, no existen estructuras importantes en otros paradigmas como las variables o los bucles, pero sí hay diferentes estrategias de ejecución de un mismo programa, funciones de orden superior, o estructuras de control como las definiciones locales.

A pesar de las ventajas de las visualizaciones a la hora de permitir manipular conceptos abstractos, y del alto grado de abstracción del paradigma de programación funcional, la aplicación de las tecnologías de visualización y animación a su enseñanza ha recibido escasa atención. Por ello, centramos nuestros esfuerzos en la utilización de las tecnologías de visualización y animación de programas en el ámbito educativo de la programación funcional.

## 1.2. Hipótesis de trabajo

Dado que la programación funcional conlleva la utilización de conceptos muy abstractos, y que las visualizaciones ayudan a formar modelos mentales de conceptos abstractos, las animaciones de programas funcionales deberían ayudar a la comprensión del paradigma funcional.

Por otro lado, se ha demostrado que la eficacia educativa de las animaciones depende en gran medida del modo en que los estudiantes las usen. Así, un enfoque constructivista [20], donde las animaciones sirvan para construir experiencias educativas, podría mejorar el aprendizaje de los conceptos animados.

Según lo dicho, la hipótesis principal que tratamos de comprobar con los trabajos descritos en esta tesis se puede enunciar como sigue:

> *La construcción de animaciones de programas funcionales mejora el aprendizaje de este paradigma de programación.*

Las formas más frecuentes en que se usan las animaciones educativas son la visión y la exploración de su comportamiento cambiando los datos de entrada. Esto se debe en parte al tardío descubrimiento de la importancia en el modo de uso de las animaciones, pero también a que una vez desarrollada la animación, permitir explorar el comportamiento del algoritmo o programa animado suele ser sencillo de desarrollar. Sin embargo, no se ha prestado apenas atención a la sencillez en el proceso de construcción de las animaciones, quizás porque cuanto más se esfuerce el estudiante más se implica, y por lo tanto más aprende.

El modelo de construcción de animaciones más común es la anotación del programa a animar con llamadas al sistema de animación. Esto requiere del estudiante varias tareas: reconocer las partes del programa que hay que visualizar, aprender un nuevo lenguaje para generar las visualizaciones, y en algunas ocasiones, incluso realizar tareas de bajo nivel como calcular la disposición de los elementos gráficos utilizados en la animación. ¿Son todas estas tareas igual de productivas en el ámbito educativo? Nosotros creemos que no. Así, de nuestra hipótesis principal podemos enunciar una primera subhipótesis relacionada con el esfuerzo dedicado al proceso de construcción:

> *Se puede diseñar un proceso de construcción sencillo y sin esfuerzo, que permita al estudiante centrarse en las tareas que realmente le ayudarán a aprender mejor los conceptos.*

Finalmente, la construcción de las animaciones es un proceso que requiere una mayor implicación por parte del estudiante que la visión. Por lo tanto, los estudiantes deberían aprender más si construyen animaciones que si sólo las ven. Así, podemos enunciar nuestra segunda subhipótesis relacionada con los efectos en el aprendizaje de los distintos grados de implicación con las animaciones:

> *El aprendizaje mejora a la par que aumenta la implicación de los estudiantes según los tres siguientes modos de uso: no usar animaciones, verlas, y construirlas.*

## 1.3. Objetivos

Como es lógico, nuestros objetivos coinciden con la comprobación de las hipótesis que hemos formulado. Por lo tanto, nuestro objetivo principal será:

> *Diseñar un modelo de construcción de animaciones de programas funcionales y evaluarlo para comprobar sus efectos en el aprendizaje de los estudiantes.*

El trabajo realizado para conseguir este objetivo no ha partido de cero. Como hemos dicho anteriormente, los investigadores llevan trabajando en este campo más 35 años. Describiremos sus desarrollos principales en el siguiente capítulo. Así, el modelo de construcción que hemos diseñado se basa en modelos existentes de ejecución, visualización y animación de programas funcionales. Describiremos en detalle este punto de partida en el capítulo 3. Siguiendo la línea de la hipótesis principal, este objetivo se puede expresar en términos de los siguientes subobjetivos:

1. Uno de los requisitos fundamentales de cualquier recurso educativo es que dirija la atención y los esfuerzos del estudiante hacia los conceptos que debe aprender, ocultando el resto de detalles que le puedan distraer. Por lo tanto, el primer subobjetivo será:

   > *Diseñar un proceso de construcción de animaciones que abstraiga al alumno de todas las tareas que no le supongan un beneficio pedagógico.*

   Para alcanzar este objetivo daremos tres pasos. En primer lugar estudiaremos procesos de construcción alternativos a la anotación del programa a animar. Una vez identificado el proceso concreto, descrito en detalle en el capítulo 3, analizaremos sus ventajas e inconvenientes. A continuación diseñaremos un nuevo proceso de construcción que aproveche las ventajas y resuelva los inconvenientes detectados. Finalmente, evaluaremos este proceso desde el punto de vista interactivo, comprobando que requiere poco esfuerzo y permite a los estudiantes centrarse en las tareas que les ayuden a aprender mejor.

2. Una vez que la construcción de animaciones se ha convertido en un proceso sencillo, hay que evaluar su eficacia pedagógica. Así, el siguiente subobjetivo será:

   > *Comprobar que el proceso de construcción de animaciones supone una mejora educativa con respecto a la visión, y ambos con respecto a la metodología clásica sin uso de animaciones.*

   Para alcanzar este objetivo realizaremos un estudio comparativo. Dentro de este estudio compararemos la visión de animaciones con su construcción. Para que dicha comparación sea significativa, mejoramos el diseño de los contenidos de las animaciones, maximizando sus posibilidades

de obtener mejoras pedagógicas. Al cambiar los contenidos de las animaciones hemos tenido que cambiar su proceso de construcción, de modo que hemos rediseñado el proceso de construcción manteniendo las propiedades de sencillez e implicación. Finalmente evaluaremos el proceso de construcción a nivel educativo en comparación con la visión de las animaciones y la metodología clásica sin uso de animaciones.

## 1.4. Aportaciones principales

Las aportaciones de esta Tesis son los resultados del trabajo realizado para alcanzar los objetivos anteriormente expuestos: proceso de construcción sencillo y sin esfuerzo que mejore el aprendizaje.

El proceso de construcción existente se basa en proveer al estudiante con el conjunto de visualizaciones de la animación en formato reducido para ofrecer una visión global. A continuación el usuario deberá seleccionar aquellas que formarán parte de la animación. Para ello también puede acceder a su versión original de forma separada. Tras analizar este proceso identificamos sus debilidades: la generación de las versiones reducidas de las visualizaciones, y la selección de las visualizaciones para formar parte de la animación. Las aportaciones persiguen la eliminación de dichas debilidades:

- Conseguir que las versiones reducidas sean lo más expresivas posible, maximizando las propiedades que mantienen de la imagen original:
  - Manteniendo la *relación de aspecto*, conseguimos que las versiones reducidas se parezcan más a las originales, ya que sus componentes internos mantienen sus posiciones relativas entre sí.
  - Resaltando la *información de la ejecución* en las visualizaciones originales conseguimos que su versión reducida mantenga dicha información.
  - Asegurando un grado de *fidelidad en la producción de la reducción* utilizando un algoritmo de reducción de imágenes de buena relación calidad/tiempo.
- Integración de la visión global –versiones reducidas de las visualizaciones– con vistas detalladas que faciliten su comprensión. Para ello hemos diseñado un técnica de visualización de información llamada *R-Zoom*. Esta técnica permite mostrar versiones detalladas de visualizaciones sin perder su contexto en forma de versiones reducidas.
- Hemos evaluado las ventajas de R-Zoom con respecto a otra alternativa, incluyendo en ambas las mejoras realizadas en la calidad de la versiones reducidas de las visualizaciones. Dicha evaluación demuestra que R-Zoom, junto con las mejoras en la calidad de las versiones reducidas, conforma un proceso de construcción de animaciones eficaz, eficiente, y satisfactorio para el usuario.
- Hemos evaluado el esfuerzo que los estudiantes deberán dedicar a construir animaciones. Concluyendo que este proceso de construcción no requiere apenas esfuerzo, permitiendo que se centren en las axctividades de aprendizaje.

A continuación hemos evaluado este proceso de construcción en el ámbito pedagógico, para ello lo hemos comparado con la visión de las animaciones, así como con un enfoque clásico sin uso de animaciones. Durante este proceso hemos hecho las siguientes aportaciones:

- Para que la evaluación sea representativa hemos decidido aumentar la calidad de las animaciones con las siguientes mejoras:
  - Integración de las animaciones con contenidos textuales que expongan el objetivo de la animación y expliquen la solución implementada en el programa que se anima.
  - Mejorar la usabilidad y facilitar la reutilización de las animaciones a nivel de usuario y plataforma.
  - Un modelo para la gestión y publicación de colecciones de estas animaciones.
- Hemos rediseñado el modelo de construcción de animaciones para contemplar las mejoras realizadas y mantener su sencillez.
- Basándonos en los resultados de la evaluación, proporcionamos un modelo de utilización de las animaciones aquí presentadas que permite obtener el máximo aprovechamiento educativo. Dicho modelo consiste en:
  - Uso de la visión de animaciones durante las explicaciones teóricas por parte del profesor.
  - Planificación de sesiones prácticas donde además de resolver ejercicios programando, se construyan animaciones de programas, dejando disponible colecciones de animaciones que se puedan consultar.
- Material didáctico desarrollado para la evaluación, en forma de animaciones de programas funcionales, que tan buen resultado han dado.

La difusión de las aportaciones realizadas en esta tesis se ha materializado en las siguientes publicaciones:

1. F. Naharro-Berrocal, C. Pareja-Flores, J. Urquiza-Fuentes, and J.Á. Velázquez-Iturbide. Approaches to comprehension-preserving graphical reduction of program visualizations. In *Proceedings of the ACM Symposium on Applied Computing 2002*, pp 771–777, New York, NY, USA, 2002. ACM Press.

2. F. Gortazar-Bellas, J. Urquiza-Fuentes, and J.Á. Velázquez-Iturbide. Reduction and flip-zooming in comprenhension-preserving miniatures of program visualizations. In *Proceedings of the Third IASTED Symposium on Visualization, Imaging and Image Processing VIIP 2003*, pp 855–861, Anaheim, CA, USA, 2003. ACTA Press.

3. M.A. Medina-Sánchez, C.A. Lázaro-Carrascosa, C. Pareja-Flores, J. Urquiza-Fuentes, and J.Á. Velázquez-Iturbide. Empirical evaluation of usability of animations in a functional programming environment. Technical report, Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 2004. Technical report 141/04.

4. J. Urquiza-Fuentes and J.Á. Velázquez-Iturbide. R-Zoom: A visualization technique for algorithm animation construction. In *Proceedings of the IADIS International Conference Applied Computing 2005*, pp 145–152, Lisboa, Portugal, 2005. IADIS Press.

5. J. Urquiza-Fuentes and J.Á. Velázquez-Iturbide. Effortless construction and management of program animations on the web. In *Advances in Web-Based Learning - ICWL 2005: Fourth International Conference*, pp 163–173, Berlin, Germany, 2005. Lecture Notes in Computer Science, vol. 3583, Springer-Verlag.

6. J. Urquiza-Fuentes, J.Á. Velázquez-Iturbide, and C.A. Lázaro-Carrascosa. Design and evaluation of R-Zoom, a new focus+context visualization technique. In *INTERACCIÓN 2006 Diseño de la Interacción Persona-Ordenador: Tendencias y Desafíos*, pp 91–100, Ciudad Real, España, 2006. Universidad de Castilla la Mancha.

7. C. Pareja-Flores, J. Urquiza-Fuentes and J.Á. Velázquez-Iturbide. WinHIPE: an IDE for functional programming based on rewriting and visualization. *ACM SIGPLAN Notices*, 42(3):14–23, Marzo 2007.

8. J. Urquiza-Fuentes and J.Á. Velázquez-Iturbide. An evaluation of the effortless approach to build algorithm animations with WinHIPE. *Electronic Notes in Theoretical Computer Science*, 178:3–13, Julio 2007.

9. J.Á. Velázquez-Iturbide, C. Pareja-Flores, and J. Urquiza-Fuentes. An approach to effortless construction of program animations. *Computers & Education*, 50(1):179–192, Enero 2008.

10. J. Urquiza-Fuentes, C.A. Lázaro-Carrascosa, and J.Á. Velázquez-Iturbide. Improving a Zoom+Pan interface with Overview+Detail or Focus+Context features: a comparative evaluation. In M.A. Redondo, C. Bravo y M. Ortega (eds.) *Engineering the User Interface*, en imprenta, Berlin Heidelberg New York. Springer-Verlag.

11. J. Urquiza-Fuentes. Evaluación de niveles de implicación de los estudiantes con la visualización de programas funcionales. In *Seminario de Investigación en Tecnologías de la Información Aplicadas a la Educación, SITIAE 2007*, en imprenta, Madrid. Dykinson.

Otras publicaciones relacionadas con el trabajo son:

- T.L. Naps, G. Rößling, P. Brusilovsky, J. English, D. Jarc, V. Karavirta, C. Leska, M. McNally, A. Moreno, R.J. Ross, and J. Urquiza-Fuentes. Development of XML-based tools to support user interaction with algorithm visualization. *SIGCSE Bulletin*, 37(4):123–138, 2005.

- T.L. Naps, G. Rößling, M.S. Hall, V. Karavirta, A. Kerren, C. Leska, A. Moreno, R. Oechsle, S.H. Rodger, J. Urquiza-Fuentes and J. Ángel Velázquez-Iturbide. Merging interactive visualizations with hypertextbooks and course management. *SIGCSE Bulletin*, 38(4):166–181, 2006.

- J.Á. Velázquez-Iturbide, D. Redondo-Martín, C. Pareja-Flores and J. Urquiza-Fuentes. An instructor's guide to design web-based algorithm animations. In *Advances in Web-Based Learning - ICWL 2007: Sixth International Conference*, en imprenta, Berlin, Germany, 2007. Lecture Notes in Computer Science, Springer-Verlag.

## 1.5. Organización de la memoria

A continuación describimos la organización del resto de la memoria, resumiendo el contenido de cada uno de los capítulos que la componen.

El **capítulo 2** describe el estado de desarrollo del campo de investigación donde se enmarcan las conclusiones principales de esta tesis. Comenzamos describiendo de forma general los conceptos de *visualización* y *visualización del software*. A continuación revisamos los trabajos más significativos en cuanto a la visualización de programas funcionales se refiere. Después exponemos los avances realizados

en el uso educativo de las visualizaciones de programas y algoritmos. Y finalmente describimos el estado actual de las visualizaciones de programas funcionales con fines educativos.

En el **capítulo 3** describimos el marco general de trabajo. Las aportaciones de esta tesis se han evaluado dentro de un entorno de programación funcional con capacidades de visualización llamado *WinHIPE*. Aquí detallamos el punto de partida de nuestro trabajo, los modelos de ejecución, visualización estática y animación implementados hasta el momento en el entorno. Y finalmente identificamos los puntos en que se ubican las aportaciones de la tesis en estos modelos.

El **capítulo 4** comprende las aportaciones relacionadas con los aspectos especialmente interactivos en el modelo de construcción de las animaciones. La idea es que la construcción de las animaciones sea simple y requiera poco esfuerzo por parte del usuario. Así, el modelo se basa en proporcionar una visión global a la par que detallada de la ejecución del programas funcionales. En primer lugar describimos las mejoras relacionadas con la visión global, basada fundamentalmente en la reducción de visualizaciones de programas. Después consideramos la integración de vistas más detallas. En primer lugar, estudiamos las técnicas de *visualización de información* que tratan este problema. A continuación describimos y evaluamos una nueva técnica llamada *R-Zoom*, que permitirá la integración de vistas más detalladas. Y finalmente analizamos el impacto de estas mejoras en el esfuerzo necesario para construir animaciones con este modelo.

El **capítulo 5** describe las animaciones como recursos educativos de entidad propia. En primer lugar se especifican los nuevos contenidos de las animaciones y su implementación como recursos utilizables vía Web. A continuación, consideramos las colecciones de estas animaciones como otro recurso educativo, proporcionando un modelo para su mantenimiento y publicación. Finalmente exponemos el uso educativo de las animaciones, así como su evaluación formal.

En el **capítulo 6** exponemos las conclusiones de nuestro trabajo, tanto desde punto de vista interactivo en la construcción de animaciones como desde el punto de vista pedagógico en su uso educativo. Para terminar identificamos líneas abiertas de investigación que dirigirán los trabajos futuros.

En los **apéndices A, B, C y D** facilitamos los detalles de los desarrollos y la experimentación realizada. Aunque la información ofrecida en capítulos anteriores es suficiente para la interpretación de los resultados, los detalles aquí expuestos permiten la reproducción de los experimentos procedimental y analíticamente.

# Capítulo 2

# Estado de la cuestión

La animación de programas forma parte del ámbito de la visualización. Para tener una visión global, en primer lugar trataremos el concepto de *Visualización*, y más en concreto la *Visualización del Software*. Después, nos centraremos en el tema de la *Visualización de la Programación Funcional*, así como el *uso educativo* de las visualizaciones de programas y algoritmos. Finalmente, analizaremos el uso educativo de las visualizaciones de programas funcionales.

## 2.1. Visualización

Existen multitud de definiciones del término *visualizar*. Según la Real Academia Española de la Lengua, visualizar es:

1. representar mediante imágenes ópticas fenómenos de otro carácter; p. ej., el curso de la fiebre o los cambios de condiciones meteorológicas mediante gráficas, los cambios de corriente eléctrica o las oscilaciones sonoras con el oscilógrafo, etc.

2. formar en la mente una imagen visual de un concepto abstracto.

3. imaginar con rasgos visibles algo que no se tiene a la vista.

4. hacer visible una imagen en un monitor.

5. hacer visible artificialmente lo que no puede verse a simple vista, como con los rayos X los cuerpos ocultos, o con el microscopio los microbios.

Así, el término *visualizar* agrupa dos conceptos diferentes. Por un lado, como Spence [209] y Ware [241] explican, hace referencia a la formación de modelos mentales. Por otro, hace referencia al uso de representaciones gráficas. Así, Tufte [225] define la *excelencia gráfica* como *"la comunicación (por medios gráficos) de ideas complejas con claridad, precisión, y eficiencia"*.

La unión del uso de representaciones gráficas y la formación de modelos mentales se puede ver en las dos siguientes definiciones de visualización:

> *"la manera visual de resolver problemas lógicos"*, Bertin [24].
>
> *"representar datos por medio del ordenador, de forma visual e interactiva, para ampliar el conocimiento"*, Card et al. [42].

La siguiente pregunta es *¿por qué visualizar?* Por un lado, Larkin y Simon [122] explican las ventajas de las representaciones diagramáticas con respecto a las textuales, describiendo cómo ayuda en los tres pasos del razonamiento: la búsqueda de datos, el reconocimiento de la información útil y la inferencia. Por otro, el sistema de visión humano es un buscador de patrones de gran potencia [241], de forma que, si se utiliza una representación visual adecuada, se puede interpretar rápidamente gran cantidad de información. Y esto ayuda a la construcción de un modelo mental sobre la entidad visualizada [209, 241].

Con una representación de la información adecuada al uso que se hará de ella, se obtienen múltiples ventajas como:

- Detectar propiedades desconocidas del objeto visualizado sin importar si dicho objeto es individual o un conjunto de ellos[1].
- Realizar un control de calidad, tanto de los datos recogidos como del proceso de recolección de esos datos.
- Facilitar la formulación de hipótesis sobre propiedades del objeto visualizado.

Dentro del término *visualización* podemos definir varios ámbitos diferentes. La *visualización científica* se centra en mostrar la apariencia visual de entidades que *sí* la tienen, pero no se ve a simple vista, p.ej., la red de vasos sanguíneos del cerebro, los flujos de líquidos, los objetos microscópicos, etc. En la visualización científica, la fidelidad al aspecto visual real es lo que importa. La *visualización de información* trata de dar apariencia visual a entidades que *no* la tienen, p.ej., mapa de la red de metro de una ciudad, gráficos de ventas de una compañía. Este tipo de visualizaciones se centra más en el uso que se va a dar a la visualización, pudiendo ocultar detalles existentes en la información.

Existen multitud de ejemplos de visualización de información, muchos de ellos comúnmente usados, como los mapas de metro de muchas ciudades donde, a pesar de no tener referencias de distancia y ubicación real de las estaciones, sí se puede obtener un modelo de recorrido de principio a fin con estaciones por las que pasar y hacer transbordos a realizar. Otro ejemplo, clásico en el campo de la visualización de información, es el conocido mapa de Minard sobre la campaña de Napoleón en Rusia (véase la figura 2.1).

El proceso de visualizar información se compone de varias fases; un fallo en alguna de ellas provocaría la desaparición de las ventajas antes mencionadas. La primera fase es la de recogida de la información y su almacenamiento. Una vez almacenada se preprocesa, de forma que sea comprensible y visualizable. A continuación, se muestra al usuario y éste forma un modelo mental de la información visualizada, utilizándolo como crea más conveniente.

Evidentemente todas las fases son importantes, pero la tercera es particularmente importante para el tema de esta tesis. En esta fase se representa gráficamente la información. La representación gráfica se hace utilizando códigos gráficos, al igual que utilizamos palabras para representar objetos de la vida real. Existen dos tipos de códigos:

**Códigos sensoriales** con los que sólo se pueden realizar tareas muy simples, pero cuya comprensión es prácticamente automática [224]. Algunos ejemplos son el color, la intensidad, la anchura, la longitud o la cercanía.

[1] En relación con la informática, se ha utilizado en técnicas de Data Warehouse y KDD (*Knowledge Discovey in Databases*)

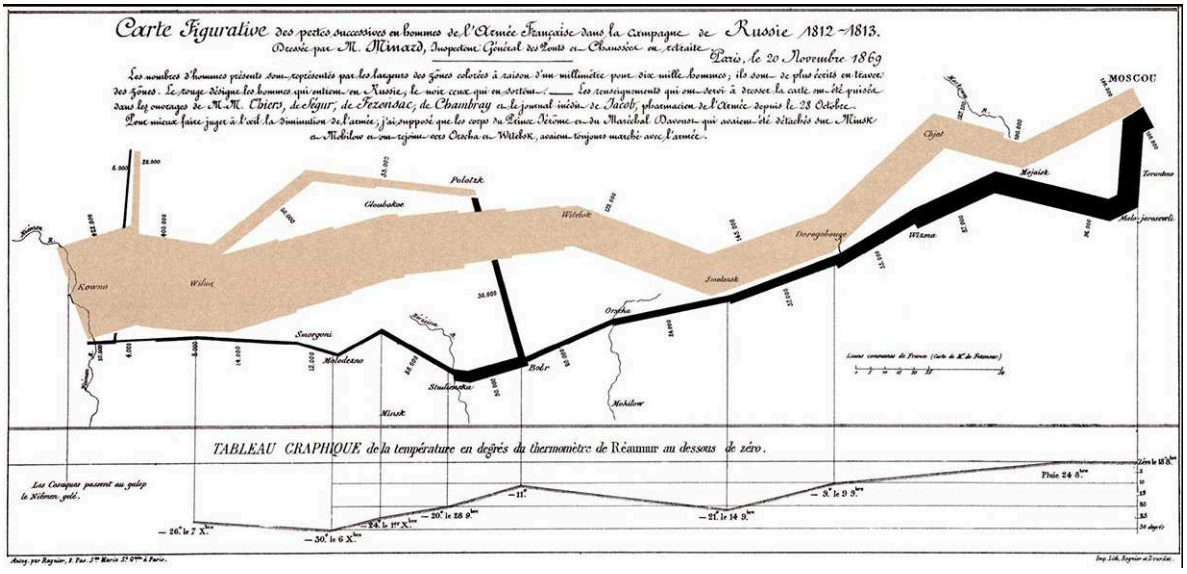



Figura 2.1: El mapa de Minard describía el avance de las tropas de Napoleón hacia la ciudad de Moscú y su retirada. En él se pueden observar distancias recorridas, sentido de avance, divisiones o agrupamientos de las tropas, número de soldados en cada lugar, fechas importantes, e incluso temperaturas soportadas (de Tufte [225])

**Códigos arbitrarios** con los que se pueden realizar tareas mucho más complejas, pero cuyo entendimiento necesita de un entrenamiento previo, además de tener gran influencia cultural [241].

Hay gran cantidad de información cuya visualización emplea códigos normalmente arbitrarios, por lo que el uso de estas visualizaciones deberían ir precedidas de cierto entrenamiento en la comprensión de esos códigos. De ahí el hecho de que una representación gráfica no siempre sea más comprensible, sino que también influirá el uso previo de los códigos gráficos utilizados así como la familiaridad con los conceptos visualizados.

Por otro lado cabe preguntarse ¿qué tipos de datos puede ser necesario visualizar? Distinguiendo entre los tipos de datos a visualizar se podrá tener una idea de sus posibles representaciones gráficas. Existen varias clasificaciones (véanse Shneiderman [203], Spence [209] o Ware [241]). En general se puede distinguir entre visualizar: entidades (objetos de interés), relaciones entre entidades, operaciones sobre entidades, atributos de entidades cuyos valores pueden expresar cantidades numéricas, datos categóricos u ordinales y finalmente entidades que en realidad son construcciones teóricas sobre las entidades, como posibles agrupaciones, clasificaciones o tendencias.

En conclusión se puede decir que una visualización adecuada de cierta información posibilita la formación de un modelo mental sobre los conceptos, su comprensión, y por lo tanto, un mejor uso de la misma. A continuación se hablará sobre cómo obtener las ventajas antes descritas cuando en vez de visualizar información se trata de *visualizar software*.

## 2.2. Visualización del software

Desde la aparición de la informática, la complejidad de los programas ha obligado a los informáticos a buscar maneras de disminuirla, aumentando la comprensión de los mismos [11]. Esa disminución de

la complejidad ha pasado por muchos estados intermedios. Algunos de ellos han sido:

- Utilización del diseño descendente, la programación estructurada y la modularidad.
- Aumento en la expresividad y claridad de los lenguajes de programación junto con las aproximaciones orientadas a objetos del diseño y desarrollo de software.
- Utilización de entornos de desarrollo integrados.
- Utilización de herramientas CASE.

Aún así no se ha conseguido eliminar toda la complejidad de los programas, y el siguiente movimiento de la informática ha sido la visualización de software. Utilizando las ideas vistas en el apartado anterior podemos adelantar que la visualización de software es la representación visual del software de forma que el programador sea capaz de obtener una mejor comprensión del mismo, muy similar a la definición dada por Domingue et al. [62] *"sistemas que usan medios visuales (y otros) para que un programador comprenda mejor el trabajo de otro (o el suyo propio)"*. De forma más detallada, Price et al. [180] definen el mismo término como *"el uso de las artes de tipografía, diseño gráfico, animación y cinematografía con las tecnologías modernas de interacción persona-ordenador e informática gráfica para facilitar tanto la comprensión humana como el uso eficaz del software de ordenador"* .

Los avances en *visualización del software* se pueden diferenciar en tres grupos:

**Presentación del código fuente** donde se empezó por aplicar técnicas de formateado[2], hasta llegar a la generación de documentación de los programas mucho más elaborada [10, 112].

**Representación del flujo de control y las estructuras de datos** donde se ha tratado la generación de diagramas de flujo a partir del código de los programas, así como la representación gráfica de estructuras de datos de alto nivel, como árboles, listas, etc.

**Animación del funcionamiento de los programas** donde se empezó mostrando el contenido de la memoria y los registros, llegándose a mostrar cambios producidos en las estructuras de datos, e incluso a niveles de abstracción mayores como las instanciaciones de objetos de determinadas clases en un programa.

Actualmente, uno de los sistemas más conocidos es BALSA [36], que permite interactuar a los usuarios con visualizaciones dinámicas de alto nivel de programas en Pascal. Sin tratar de ser exhaustivos, pues existen gran cantidad de sistemas, podemos citar a otros como ZEUS [34] que es un sistema de visualización de algoritmos para algunos dominios como geometría computacional o sistemas operativos. Para programación paralela se tienen los sistemas de visualización Hence [19], XPVM [113], ParaGraph [85], IVE [121], PIE [126], PAVANE [190] o POLKA [216]. Como herramientas para C++ existe la de Cheng y Gray [46], para Java JIVE [73] y para PROLOG están la de Lazzeri [125] y TPM [64].

Aún después de definir la visualización del software, resulta bastante difusa la relación que existe entre las dos palabras. Por ejemplo, tanto mostrar el diagrama de flujo de un algoritmo como los cambios ocurridos en la pila del sistema durante la ejecución de un programa son visualizaciones posibles, pero está claro que no pertenecen a la misma categoría. De hecho, el diagrama de flujo es lo que se llama una *visualización estática*, mientras que la secuencia de cambios de la pila del sistema es *una visualización dinámica*, también llamada animación.

[2] También conocidas como "pretty-printing".

Otra diferencia existe en el nivel de abstracción que se visualiza. Un nivel alto se usa en la *visualización de algoritmos*, definida por Price et al. [180] como *"visualizaciones de alto nivel que describen el software"*. Un nivel bajo se referiría a la visualización de la implementación de un algoritmo en un determinado lenguaje de programación, también llamada *visualización de programas*[3]. Roman y Cox [188] la definieron como *"la presentación gráfica, monitorización y exploración de programas expresados de forma textual"*.

Finalmente queda por ver, dentro del software o del programa, ¿qué interesa visualizar? y ¿para qué se puede usar esa visualización? En cuanto al *qué*, existen varias posibilidades: el código, o los datos, o ambos. Puede que sólo interese visualizar el código de un algoritmo y podríamos estar ante un diagrama de flujo, o los datos y podríamos ver la animación de los cambios producidos por un algoritmo de ordenación en una lista de elementos.

En cuanto al *para qué*, existen varios puntos diferenciados:

- Exhibición del comportamiento de un programa, normalmente con fines pedagógicos. En este punto es particularmente útil la animación de programas y algoritmos.
- Depuración lógica. Permitiendo comprender el funcionamiento del programa y comprobar si es correcto o no.
- Depuración de rendimiento. Stasko y Wehrli [211] acuñan el término *visualización de la computación* para este punto.

Qué interesa visualizar, cómo se visualiza, cómo se obtiene la información a visualizar, cómo se genera la visualización, para qué se usará la visualización, junto con otros muchos aspectos son criterios que ayudan a clasificar un sistema de visualización de software. Existen varias taxonomías sobre sistemas de visualización del software: Oudshoorn et al. [167], Price et al. [180], Myers [143], Brown [35, 33], Stasko and Patterson [217], o Roman y Cox [189]. Todas ellas caracterizan los sistemas de visualización del software desde múltiples puntos de vista. Nosotros nos centraremos en una parte de estos sistemas, los sistemas de visualización con fines educativos que utilizan el paradigma funcional.

## 2.3. Visualización de la programación funcional

Los paradigmas de programación más conocidos son el imperativo y el orientado a objetos, pero existen otros como el lógico o el funcional. Todos ellos tienen características que les diferencian y que influyen en su visualización. Por ejemplo, en programación funcional no tiene sentido hablar de variables ya que no existen. Otra diferencia existente entre los distintos paradigmas es su modelo de ejecución. Dependiendo de este se diseñará tanto la forma de generar animaciones como el formato gráfico de las visualizaciones de estos programas.

En esta sección hacemos una revisión de los sistemas de programación funcional que, de una forma u otra, utilizan técnicas de visualización para mostrar algún aspecto del funcionamiento de los programas funcionales. En primer lugar haremos una exposición de las características particulares del paradigma funcional, luego daremos una breve descripción de cada sistema, y finalmente estudiaremos cómo se ha abordado la visualización de las características particulares del paradigma funcional en los sistemas descritos.

[3]No confundir con la programación visual, que trata de la especificación gráfica de programas de ordenador [188].

### 2.3.1. Características del paradigma funcional

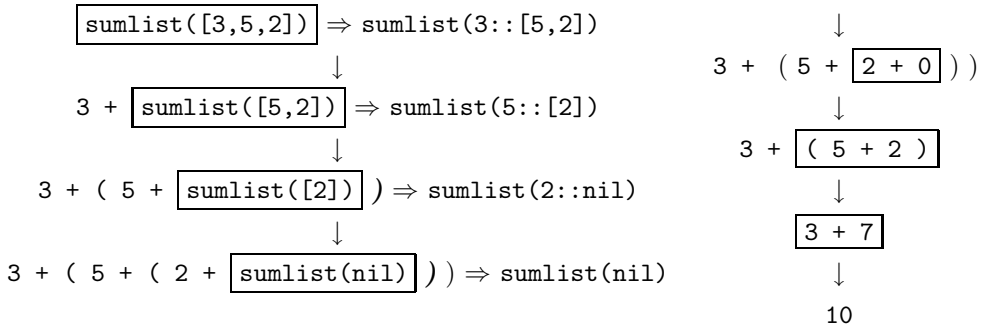
El paradigma de programación funcional tiene varias características que necesitan visualizarse para comprender mejor la ejecución de un programa. Un programa funcional es un conjunto de declaraciones, fundamentalmente de tipos y funciones. Las funciones se declaran definiendo su dominio, rango y transformación. Una transformación puede estar compuesta de varias cláusulas, o reglas, donde se especifica en forma de ecuación la correspondencia entre los valores del dominio y del rango. Sirva como ejemplo el siguiente programa HOPE [171] que calcula la suma los elementos de una lista.

```
dec sumlist: list(num) -> num;
--- sumlist nil <= 0
--- sumlist (cabecera :: resto) <= cabecera + sumlist resto;
```

La primera línea del programa define el tipo de la función con el dominio, listas de enteros `list(num)`, y el rango, enteros `num`. Las dos siguientes líneas definen la transformación con dos reglas, una para las listas vacías, y otra para el resto de listas.

La ejecución de un programa funcional es la aplicación a una expresión inicial de sucesivas transformaciones, hasta obtener una expresión en *forma normal*, que es aquella sobre la que no es posible aplicar más transformaciones. Este proceso de transformación de la expresión suele llamarse *proceso de reescritura*. La expresión está formada operadores y operandos, ambos pueden estar predefinidos en el lenguaje o ser definidos por el programador. Cada transformación, o paso de reescritura, es el resultado de aplicar un operador a sus operandos. Cuando una función se compone de varias reglas de transformación, la selección de la regla a ejecutar se realiza mediante un proceso de ajuste de patrones según los parámetros de la llamada.

El proceso de transformación de la expresión inicial en su forma normal también se llama *evaluación* de la expresión. A continuación mostramos la evaluación paso a paso de la expresión `sumlist([3,5,2])`, donde se pueden observar todos los pasos de reescritura realizados. Las subexpresiones encuadradas indican la parte de la expresión que se va a reescribir en el siguiente paso, también llamada *redex*. A la derecha de cada paso que usa una función, detallamos el ajuste de patrón necesario para seleccionar la regla de transformación que hay que aplicar.




Como se ve en el ejemplo anterior, los pasos de reescritura se aplican a diferentes partes de la expresión a evaluar. En cada paso de reescritura se evalúa una (sub)expresión, dando lugar a un (sub)resultado. Por lo tanto, parece relevante visualizar tanto la evaluación de las (sub)expresiones como sus correspondientes resultados obtenidos.

Otra característica a visualizar es el orden en que se ejecutan las llamadas a las funciones. Dos detalles importantes son la evaluación de los parámetros y el uso de ajuste de patrones para seleccionar la regla de transformación apropiada.

El entorno de los nombres utilizados en las funciones asigna sus valores; por lo tanto su visualización es también interesante. Más aún, si se está trabajando con estructuras de datos complejas, como listas o árboles, sería deseable tener visualizaciones especiales para ellos.

El orden de ejecución de un programa funcional viene fijado por la elección del siguiente redex a reescribir. Existen varias estrategias de selección del redex; las dos más importantes son la estrategia *impaciente*, utilizada en el ejemplo anterior, y la estrategia *perezosa*, que retrasa al máximo la evaluación de cualquier subexpresión. A continuación mostramos un ejemplo con la típica función del cálculo del factorial de un número:

```
dec factorial: num -> num;
--- factorial n <= if n = 0 then 1 else n * factorial(n-1);
```

La ejecución de la expresión `factorial(3+1)` utilizando la estrategia de evaluación impaciente empezaría de la siguiente forma:
`factorial(` `3+1` `)` → `factorial(4)` → ...
Mientras que la ejecución de la misma expresión utilizando la estrategia de evaluación perezosa empezaría de la siguiente forma :
`factorial(3+1)` → `if` `(3+1)` `= 0 then 1 else` `(3+1)` `* factorial(` `(3+1)` `-1)` → ...

La función `factorial` se aplica antes de evaluar su parámetro de llamada `3+1`. Aunque en la expresión resultante, la subexpresión `(3+1)` aparece tres veces, realmente se almacenará y evaluará una sola vez. Esto se llama compartición de expresiones, y es un concepto suficientemente importante como para visualizarlo en el caso de las evaluaciones perezosas.

### 2.3.2. Sistemas de visualización de la programación funcional

Normalmente, las características visualizadas y la forma en que se lleva a cabo la visualización dependen del tipo de sistema que se trate. Nosotros hemos clasificado los sistemas en cuatro categorías: entornos integrados de programación, depuradores, sistemas de visualización y sistemas para la enseñanza. El criterio de clasificación está principalmente guiado por el ánimo de los autores de cada uno de los sistemas.

Los *entornos integrados de programación* suelen agrupar un conjunto de herramientas bajo la misma interfaz general. *CIDER*[4] [84] usa el lenguaje perezoso Curry [3]. Además integra herramientas de edición y análisis de programas, un depurador gráfico y una herramienta que muestra el árbol de llamadas. La información sobre la ejecución de los programas la recoge en una traza. Su depurador permite inserción de puntos de control, así como el cambio en el sentido de la ejecución.

*ZStep* [129] es un entorno integrado para Lisp [166]. Su depurador permite la ejecución en ambos sentidos, evaluación de expresiones seleccionadas, ejecución completa y control de velocidad de ejecución. También proporciona el árbol de llamadas, y de nuevo la información de ejecución se guarda en una traza. *ZStep* muestra simultáneamente el código fuente y su ejecución. Los errores producidos durante la ejecución los sitúa en el mismo punto en el que debieran aparecer los valores correctos, de forma que es posible seguir la traza de los valores erróneos.

El último entorno integrado es *TERSE* [108]. Aunque no es un entorno de programación funcional lo tratamos como tal. En realidad es un sistema de programación por reescritura (que es la base de la

[4]Las imágenes del sistema *CIDER* utilizadas en este capítulo son cortesía de Michael Hanus, [84]

ejecución de la programación funcional). Se ha desarrollado en Standard ML/NJ[5] y permite traducir sus programas a Standard ML [138]. Durante la ejecución, permite seleccionar de entre todos los posibles redex aquel que se va a reescribir a continuación, ya sea mediante su selección individual, o en función de una estrategia de ejecución determinada. También permite escoger la regla a ejecutar. Permite tener la visión global de toda la expresión, representada como un árbol, así como una visión detallada de un área en particular. Finalmente, genera la secuencia de expresiones intermedias producidas durante la ejecución.

Hemos estudiado seis *depuradores*. *Freja* [163] y *Buddha* [178] usan subconjuntos del lenguaje Haskell [104]. Son depuradores algorítmicos y usan el grafo de dependencia de reescrituras para guiar al usuario durante la depuración.

*Hat* [208] también está diseñado para un subconjunto de Haskell, y genera una traza con los redex evaluados que se puede explorar de forma gráfica. Un sistema similar es el desarrollado por Watson y Salzman [242], que aquí llamaremos *TBL*[6] (*Trace Browser for a Lazy functional language*). TBL utiliza un lenguaje funcional perezoso desarrollado por los autores, y además de permitir la navegación por la traza de redex, muestra de forma paralela el código fuente del programa.

*Hood* [74], sin embargo, permite utilizar todas las características del lenguaje Haskell. Mediante la modificación del código fuente, se insertan llamadas al sistema de visualización (también llamadas "observaciones") allá donde se quiera obtener la visualización (tanto en una función como en una estructura de datos). Las visualizaciones se obtienen como resultado de la ejecución del programa. Esta misma idea se implementa en *COOSy* [30], desarrollado para el lenguaje Curry.

*Prospero* [221] y *Hint*[7] [68] son muy similares. Mientras que *Prospero* usa el lenguaje Miranda [226], *Hint* usa un subconjunto de Haskell. Ambos usan la traza de ejecución y permiten la inserción de puntos de control, pero no permiten la marcha atrás durante la ejecución.

Hemos encontrado dos *sistemas de visualización*. *GHood* [182] genera representaciones gráficas de las visualizaciones textuales de Hood. Tiene la posibilidad de generar los gráficos en formato EPS, también proporciona una interfaz típica de reproducción de vídeo, donde se puede controlar la velocidad de reproducción. *Visual Miranda*[8] [6] usa el lenguaje Miranda y genera una traza textual que luego muestra de forma gráfica.

Los *sistemas para la enseñanza* suelen centrarse en aspectos particulares para los que están especialmente diseñados. *Evaltrace* [223] realmente es una notación para construir visualizaciones de programas Lisp. Las visualizaciones son documentos generados con LaTeX. Se centra especialmente en diferenciar las acciones de evaluación y aplicación, aunque también visualiza macros y efectos laterales.

*KIEL*[9] [23] está diseñado para un subconjunto del lenguaje Standard ML donde sólo se trabaja con funciones de primer orden. Permite cambiar la estrategia de ejecución, así como ejecutar un número determinado de pasos de reescritura. Sus visualizaciones son la representación gráfica del árbol de sintaxis abstracta.

*DrScheme* [67] utiliza el lenguaje Scheme [181]. Está diseñado para trabajar con cuatro subconjuntos diferentes del lenguaje. Cuando se produce un error, es capaz de localizar la llamada que lo produjo. También posee un depurador estático, que mediante inferencia de tipos puede predecir errores.

*KAESTLE & FooScape* [15] son dos herramientas diseñadas para el lenguaje Lisp, siendo capaces

[5] Entorno de desarrollo para Standard ML, con librerías asociadas. http://www.smlnj.org/

[6] Las imágenes del sistema *TBL* utilizadas en este capítulo son cortesía de Richard Watson, [242]

[7] Las imágenes del sistema *Hint* utilizadas en este capítulo son cortesía de Colin Runciman, [68]

[8] Las imágenes del sistema *Visual Miranda* utilizadas en este capítulo son cortesía de IEEE y Mikhail Auguston, [6]

[9] Las imágenes del sistema *KIEL* utilizadas en este capítulo son cortesía de Rudolf Berghammer, [23]

de visualizar únicamente estructuras de datos y grafos de llamadas a las funciones. Pueden generan tanto fotogramas de cada visualización como secuencias de ellos.

### 2.3.3. Visualización de las Características del Paradigma Funcional

A continuación detallamos cómo los sistemas descritos anteriormente visualizan las características particulares del paradigma de programación funcional.

#### (Sub)expresiones, redex y (sub)resultados

Las expresiones funcionales son estructuras inherentemente jerárquicas, de forma que todos los sistemas utilizan árboles para su representación interna (con la excepción de los lenguajes perezosos donde se utilizan grafos dirigidos). Así, bastantes sistemas también representan gráficamente a las expresiones como árboles (véase figura 2.2): *CIDER*, *KIEL* (que además permite interacción directa con el árbol de sintaxis abstracta), *Prospero*, *Hint* y *TERSE* (que representa de forma diferenciada constructores, nombres de valores y funciones).

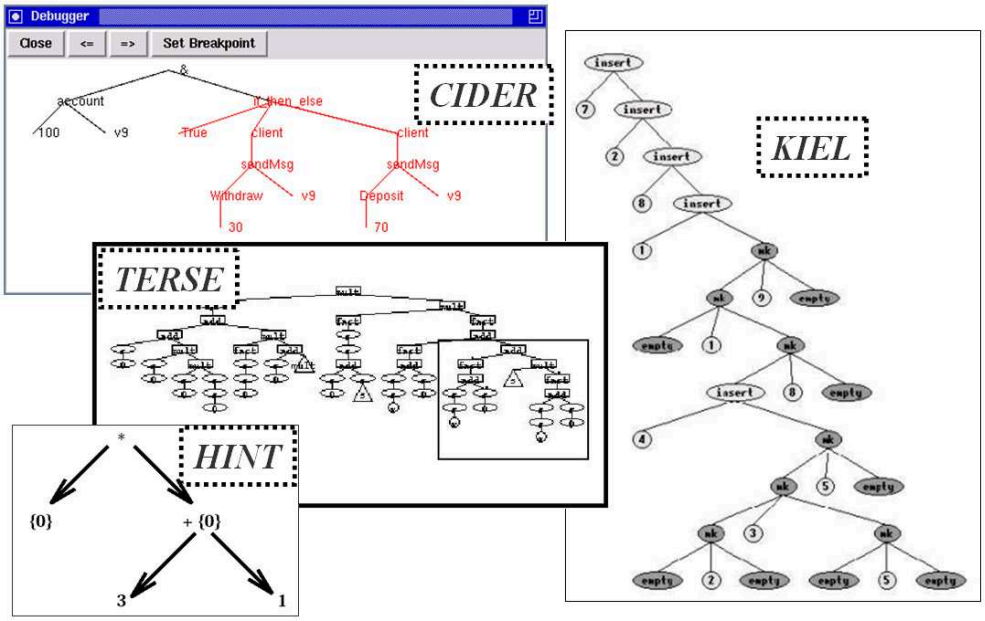



Figura 2.2: Representación gráfica de expresiones como árboles en los sistemas *CIDER*, *KIEL*, *Hint* y *TERSE*([108] ©1994 ACM)

Cuando una expresión es grande, su visualización puede llegar a confundir más que aclarar, por eso algunas herramientas ofrecen la posibilidad de reducir de alguna forma estas expresiones (véase figura 2.3). *Prospero* y *Hint* permiten aplicar filtros espaciales y temporales; *Hint* además proporciona un metalenguaje para especificar dichos filtros. *TERSE* resume subárboles en nodos con una representación especial diferente al resto de nodos. *ZStep* permite filtrar expresiones definiendo condiciones. Finalmente, *Evaltrace* representa de forma más compacta los pasos de evaluación triviales.

Todos los sistemas excepto COOSy, Hood, GHood, y Evaltrace destacan el redex de alguna forma (véase la figura 2.4). El caso de COOSy, Hood, y GHood es especial ya que el programador selecciona tanto los elementos a visualizar como el lugar del código fuente donde le interesa hacer la visualización.

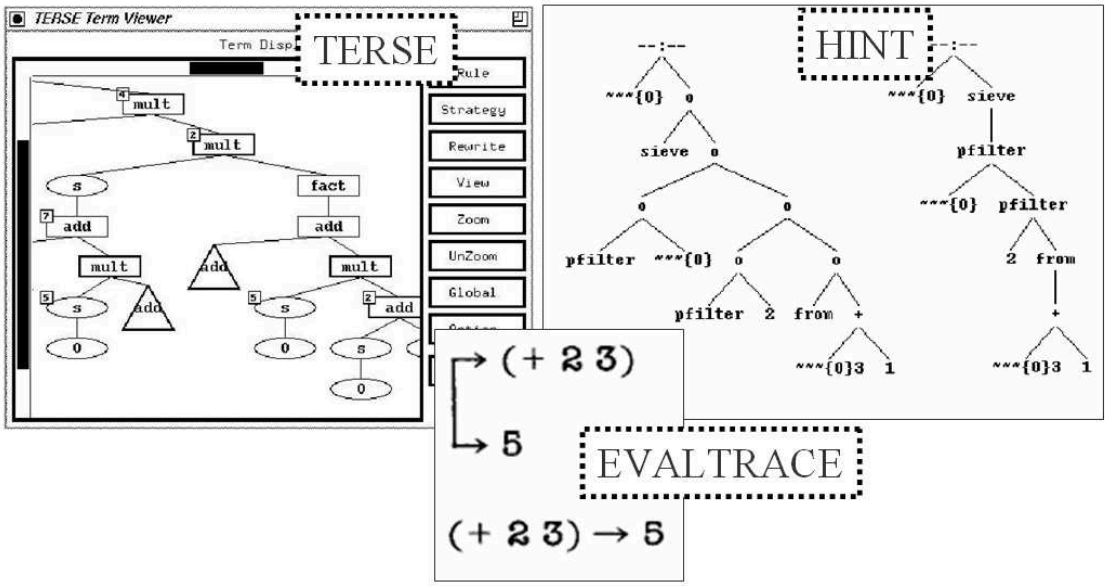



Figura 2.3: Representación más compacta de expresiones grandes con *Hint*, *Evaltrace* ([223] ©1992 ACM) y *TERSE* ([108] ©1994 ACM)

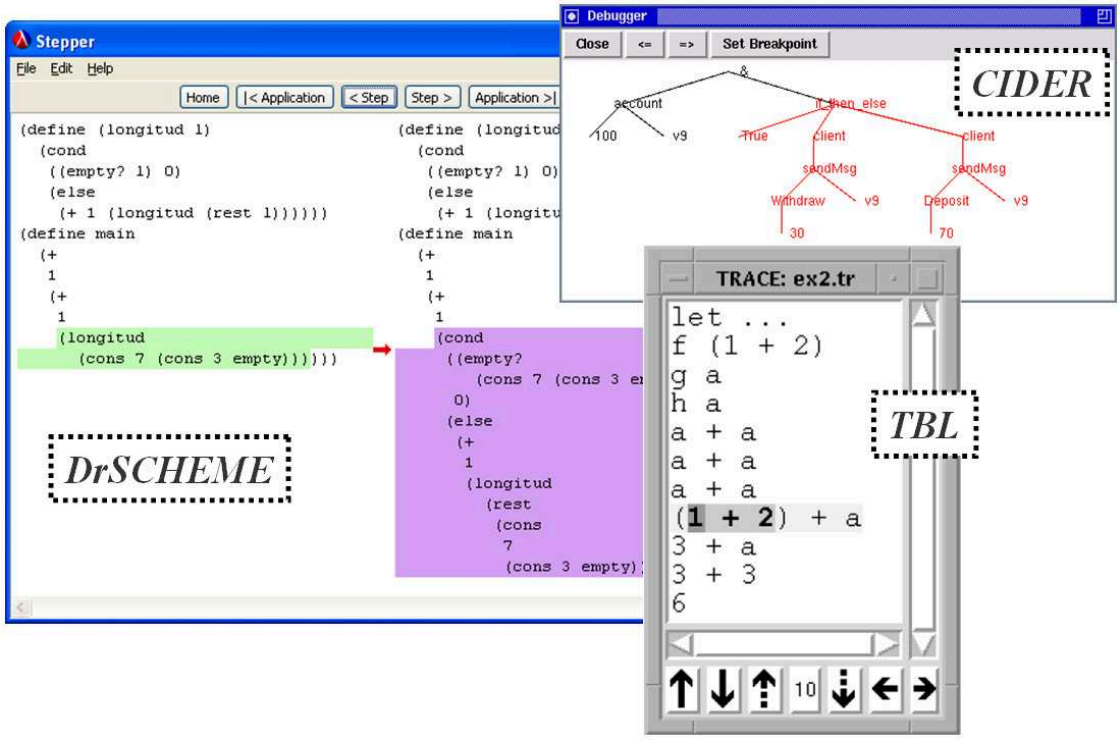



Figura 2.4: Resaltando el redex dentro de las expresiones con *CIDER*, *TBL* y *DrScheme*

*Hat*, *Freja*, *Buddha* y *Evaltrace* conectan las subexpresiones y el resultado de su evaluación. *DrScheme* muestra simultáneamente la expresión actual, su evaluación, y la definición de la función usada en el paso de reescritura. *TBL* destaca el resultado de la reescritura anterior, y el redex del estado actual junto con la parte del código fuente que se está ejecutando en este momento. *Visual Miranda* establece una conexión visual entre la expresión, sus subexpresiones y el resultado final (véase la figura 2.5).

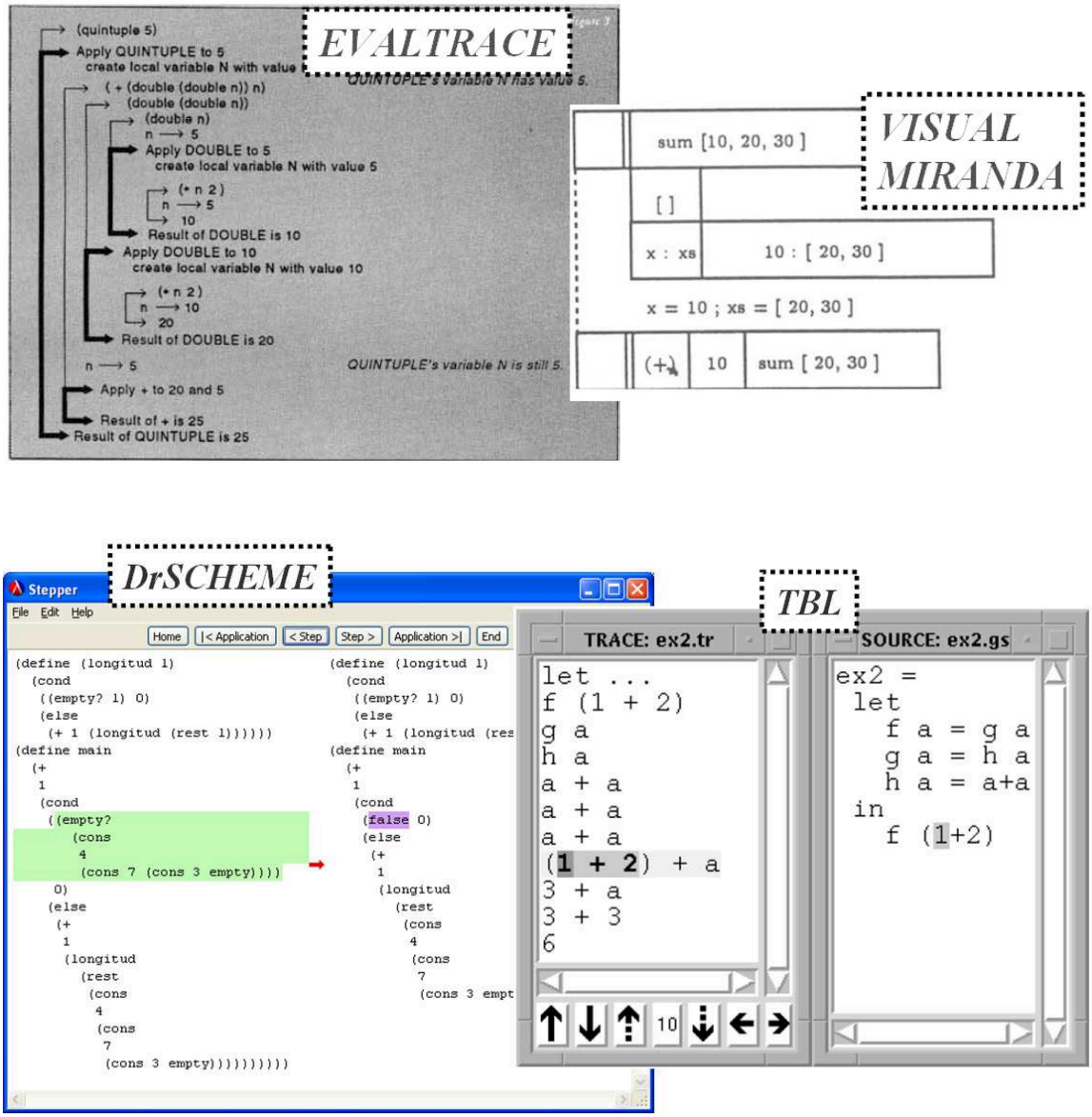



Figura 2.5: Conexión visual entre las subexpresiones y sus resultados con *Evaltrace* ([223] ©1992 ACM), *Visual Miranda* ([6] ©1994 IEEE), *DrScheme* y *TBL*

**Llamadas, aplicaciones de funciones y ajuste de patrones**

*KAESTLE & FooScape* visualizan las llamadas a funciones mediante un diagrama estático, donde las funciones se representan como elipses y las llamadas como arcos dirigidos desde la función *llamante* hacia la función *llamada* (véase la figura 2.6). El usuario puede ocultar las llamadas realizadas desde alguna función, reduciendo así el contenido del diagrama. Finalmente, permite una visualización dinámica, ya que puede destacar aquellas funciones cuya ejecución no ha terminado.

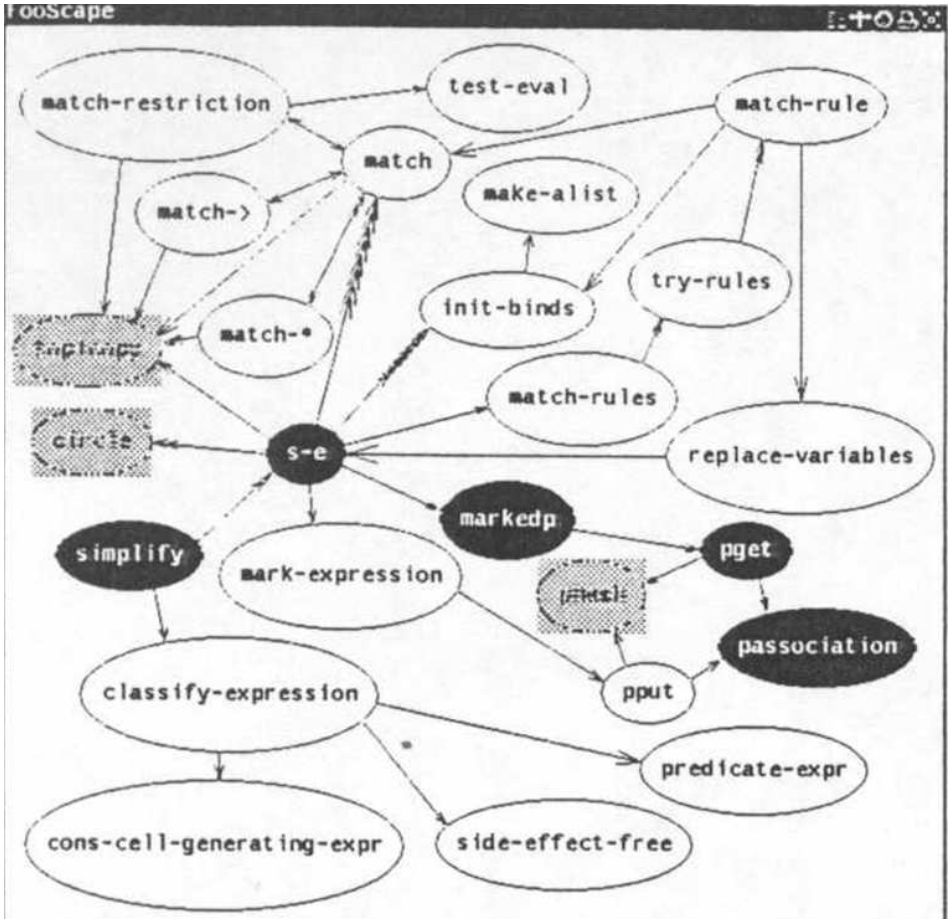



Figura 2.6: Visualización de llamadas a funciones de *KAESTLE & FooScape* ([15] ©1986 ACM)

*Evaltrace* se centra en diferenciar claramente entre la evaluación de argumentos, y la aplicación de la función a los mismos. Esta distinción se lleva a cabo utilizando distintos tipos de líneas: cuando se evalúan los parámetros de llamada se utiliza una línea de trazo fino, mientras que cuando se aplica la función se utiliza una línea de trazo mucho más grueso (véase la figura 2.5). *Visual Miranda* muestra el proceso de ajuste de patrones al completo (véase la figura 2.7). *TBL* es capaz de mostrar los distintos patrones seleccionados ya que muestra el código fuente en ejecución resaltado.

Finalmente, *Hood* y *COOSy* permiten visualizar las llamadas a las funciones, si la llamada al sistema de visualización se sitúa en la definición de la propia función (véase la figura 2.8).

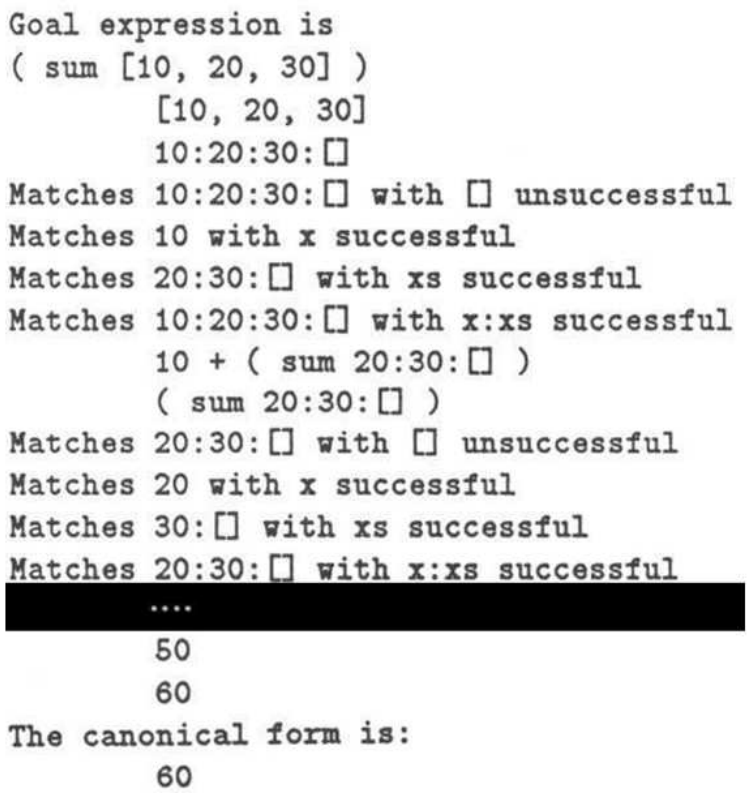



Figura 2.7: Visualización del ajuste de patrones con *Visual Miranda* ([6] ©1994 IEEE)

```
ultimo = observe "ultimo" ultimo'
ultimo' (x:xs) = ultimo xs
ultimo' [x] = x
```

```
--- ultimo
    { \ (_ : _ : []) -> throw <Exception>
    , \  ( _ : []) -> throw <Exception>
    , \  [] -> throw <Exception>
    }
```

Figura 2.8: *Observación* de Hood de una llamada a una función

### Valores de nombres y estructuras de datos

El entorno de los nombres y sus valores se visualiza de múltiples formas, pero todos los sistemas muestran el valor de los nombres. Hood y COOSy representan un caso especial, ya que muestran el valor de un nombre, en aquellos puntos del programa donde se ha ubicado la llamada al sistema de visualización. *Evaltrace* identifica un entorno conectando su comienzo y fin con una línea gruesa. Si la línea es de trazo continuo, entonces este entorno es hijo del entorno global. En caso contrario, los nombres son locales y el entorno es hijo del inmediatamente anterior a él (véase la figura 2.9). *DrScheme* conecta los nombres y sus valores mediante líneas (véase la figura 2.9). Al mostrar la evaluación y el código fuente simultáneamente, *TBL* conecta visualmente los nombres y sus valores, pero sólo en el momento en que se evalúan (véase la figura 2.5). Finalmente, *Visual Miranda* muestra el valor de cada nombre antes de evaluar una expresión (véase la figura 2.5).

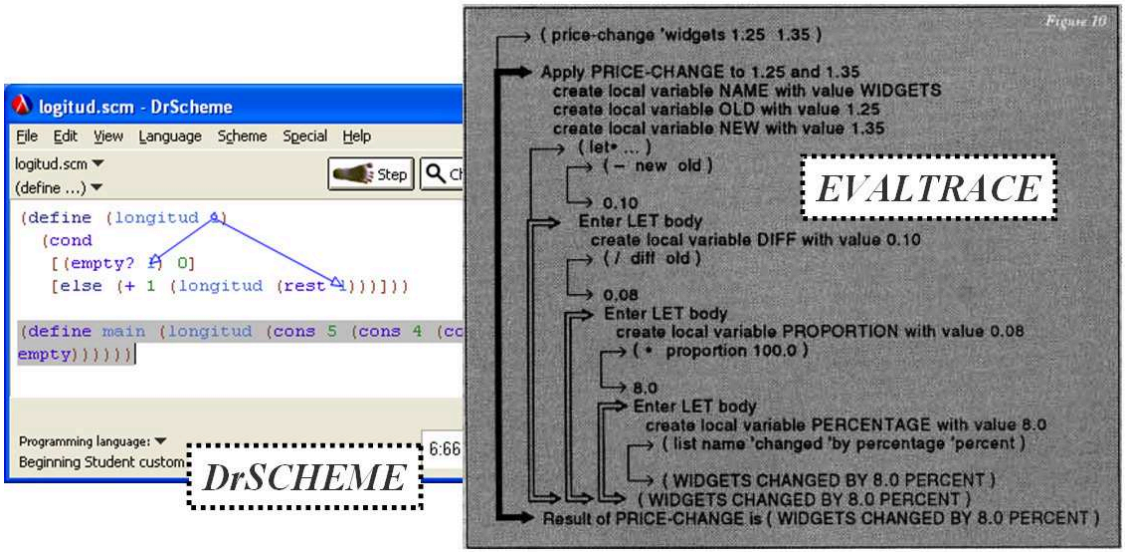



Figura 2.9: Visualización de entornos, nombres y valores con *Evaltrace* ([223] ©1992 ACM) y *DrScheme*

Sólo un sistema ofrece visualizaciones alternativas para las estructuras de datos complejas, *KAESTLE & FooScape* (véase la figura 2.10). Así, visualiza listas dibujando a sus elementos dentro de rectángulos, colocándolos consecutivamente o enlazándolos con flechas, según sea necesario. Además, permite tanto la modificación del diseño gráfico de cada lista como de sus contenidos.

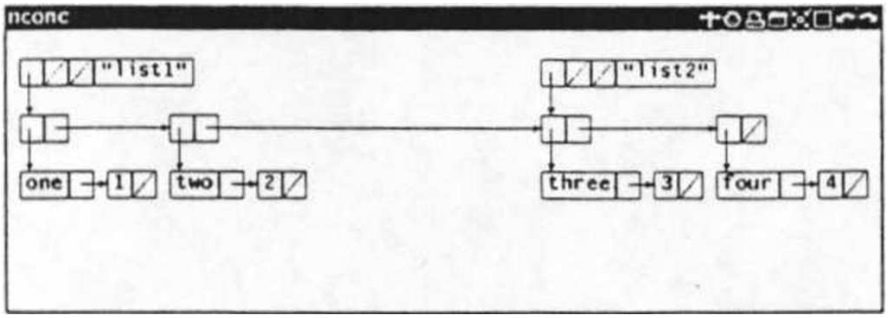



Figura 2.10: Representación gráfica de listas con *KAESTLE & FooScape*. [15] ©1986 ACM

**Estrategia de Evaluación Perezosa**

Los sistemas que trabajan con evaluación perezosa deberían visualizar la compartición de subexpresiones. *CIDER* y *Hat* no visualizan las subexpresiones compartidas hasta que no son reescritas, en ese momento destacan todas las apariciones y aplican la transformación a todas ellas. *Prospero* y *Hint* (véase la figura 2.11) muestran la subexpresión compartida al completo una sola vez, y luego conectan el resto de apariciones de esta con flechas o etiquetas. *TBL* no visualiza de forma especial las expresiones compartidas, pero sí destaca cuándo una subexpresión no está evaluada, mostrando dicha subexpresión en estilo cursiva.

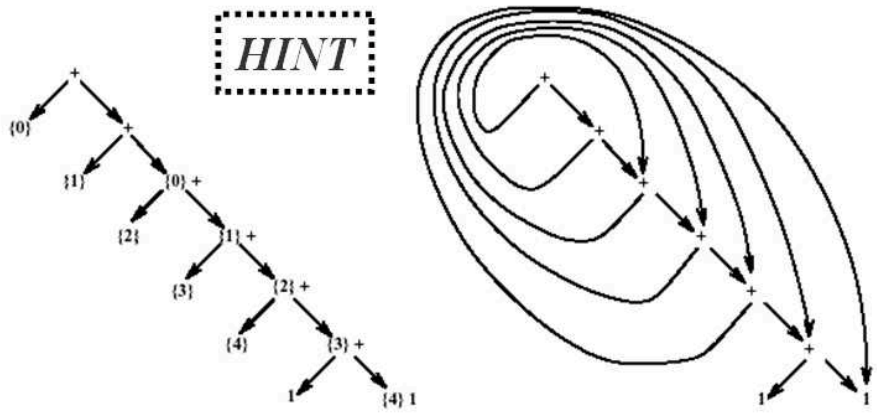



Figura 2.11: Visualización de compartición de subexpresiones con *Hint*

Hasta ahora nos hemos centrado en los aspectos tecnológicos relacionados con la visualización de programas en el paradigma funcional. En el resto del capítulo trataremos los aspectos educativos de la visualización de programas y algoritmos.

## 2.4. Visualización de programas y algoritmos con fines educativos

Si la visualización ayuda a formar modelos mentales de conceptos abstractos, y los conceptos explicados en cualquier asignatura de programación son abstractos, parece lógico intentar utilizar la *Visualización del Software* para enseñar programación. Baecker [9] fue un de los pioneros en este campo animando un conjunto de algoritmos de ordenación. Aunque sea poco riguroso, en principio, para cualquier estudiante es más atractivo poder ver el funcionamiento de un algoritmo que atender a una explicación de un profesor. Por otro lado, no estamos hablando de sustituir al profesor, sino de proporcionar nuevos medios que ayuden a los alumnos en el aprendizaje de las materias. Nótese que se ha dicho atractivo y no productivo: la obtención de evidencias empíricas sobre la eficacia de las visualizaciones en la enseñanza de conceptos informáticos no es tarea fácil ni inmediata [133].

Aún así, Kehoe et al. [109] ofrecen resultados empíricos donde se muestra una mejora de los estudiantes en su progreso personal, aunque no de resultados finales. Por lo tanto se puede intentar usar la *Visualización del Software* para facilitar la construcción de modelos mentales que ayuden al estudiante en el aprendizaje de la programación o asignaturas similares. Se podrían mencionar diferentes ventajas

del uso de la *Visualización del Software* en la enseñanza de programación, tanto desde el punto de vista del alumno, como del profesor:

- Las conceptos abstractos visualizados se asimilan más fácilmente.
- La visualización aumenta la motivación.
- El profesor puede elegir qué aspectos de la materia hay que mostrar en la visualización, resaltando aquellos más importantes para el objetivo final.
- Ayuda a la explicación del comportamiento dinámico de programas y algoritmos, permitiendo la detección de problemas y errores.
- El ritmo de la visualización se puede adaptar al ritmo de aprendizaje de los alumnos.
- Ofreciendo distintas vistas del funcionamiento de un algoritmo, los alumnos pueden descubrir ellos mismos características del algoritmo.

Hasta ahora tan sólo se han mostrado las ventajas de utilizar visualizaciones, tomando como dogma de fe cuando no lo es la frase: "*una imagen vale más que mil palabras*". La mayoría de los autores que han trabajado con *Visualización del Software* en la enseñanza (véanse [4, 9, 14, 21, 76, 109, 142]) concluyen que la eficacia de su uso depende del cumplimiento de ciertas condiciones como:

**Explicaciones añadidas a la visualización.** El hecho de ver una visualización no implica su comprensión. Hay que explicar cómo funciona la visualización, qué muestra, cuál es su objetivo y cómo lo conseguirá.

**Capacidad de abstracción.** La visualización debe resaltar la información que atañe a su objetivo, ocultando aquellos detalles que puedan confundir.

**Diseño gráfico apropiado.** El diseño gráfico debe ser atractivo y claro. Si los convenios visuales (o códigos arbitrarios, [241]) no son conocidos por los estudiantes, se deben explicar previamente. La idea es minimizar el tiempo invertido en entender la visualización frente al tiempo invertido en comprender el algoritmo.

**Complejidad variable.** La complejidad de lo que se muestra se debería poder ajustar. Las visualizaciones para la primera vez que se explica un algoritmo no son las mismas que aquellas que muestran posibles optimizaciones o casos particulares de su funcionamiento.

**Temporización y vuelta atrás.** Tratándose de animaciones (visualizaciones dinámicas), la temporización es fundamental: la velocidad de la visualización al principio será mucho menor que al final. Al principio cuesta entender el algoritmo, pero una vez comprendido, puede incluso aburrir al alumno. También es importante ofrecer la posibilidad de cambiar el sentido de ejecución del programa o algoritmo, pudiendo volver a ver la visualización de un momento de la ejecución anterior al visualizado actualmente.

**Multiplicidad de vistas.** Es conveniente poder ofrecer al usuario de la visualización varias vistas de forma que pueda elegir cuál le interesa más. Por ejemplo, en un algoritmo de ordenación podría interesar tanto la línea del algoritmo que se está ejecutando en ese momento, como el estado del vector que hay que ordenar.

**Integración con el entorno de programación.** La generación de una visualización podría ser una acción más de las posibles, como ejecutar, compilar o depurar un programa, permitiendo al alumno experimentar con el sistema creando sus propias visualizaciones.

### 2.4.1. Factores a tener en cuenta en el uso educativo de las visualizaciones

De forma intuitiva, el uso educativo de las visualizaciones se asimila a la observación por parte del estudiante de representaciones gráficas que muestran el funcionamiento de un programa o algoritmo. Sin embargo, las experiencias de este tipo que han registrado mejoras en el aprendizaje de los estudiantes son anecdóticas. Un ejemplo puede ser el experimento realizado por Saraiya et al. [197] donde se obtuvieron mejoras si los alumnos eran capaces de controlar el ritmo la animación. Además, en este experimento no se aisló en su totalidad el factor de la visualización, ya que los materiales de los alumnos que usaron las visualizaciones eran de mayor calidad.

Por el contrario, otras experiencias iniciales mostraron la complejidad de obtener resultados pedagógicos positivos con las animaciones. Así, Stasko et al. [214] comprobó que la simple visión de animaciones, en comparación con materiales textuales de la misma calidad, no aporta mejoras en el aprendizaje de los estudiantes. Dando un paso más, Jarc y Feldman [99] y Byrne et al. [39] probaron escenarios distintos como la realización de tareas de predicción sobre el comportamiento de los algoritmos, pero tampoco obtuvieron resultados significativos a favor del uso de las animaciones.

Así, empezaron a considerarse escenarios de uso de las animaciones más *activos*, como la exploración del comportamiento de los algoritmos o la construcción explícita de animaciones por parte de los propios alumnos. Estos escenarios dieron resultados más alentadores [110, 124, 213], aunque su representatividad fuera pequeña. Hundhausen et al. [94] llevaron a cabo un estudio mucho más general y riguroso, con resultados significativamente favorables para el uso de las animaciones. Este trabajo ha supuesto un punto de inflexión en la investigación del uso de animaciones de algoritmos con fines educativos. Tras hacer una revisión de 24 experimentos sobre la eficacia pedagógica del uso de las animaciones, llegaron a la conclusión de que *"lo realmente importante es lo que el alumno hace con la visualización y no lo que ve en ella"*.

Así, se pueden encontrar varios trabajos que estudian las distintas formas de interacción de los alumnos con las visualizaciones. Diehl y Kerren [61] describieron distintas formas de interactuar con visualizaciones de autómatas y máquinas abstractas. En concreto identificaron cuatro *niveles de exploración*:

**Estático** que permite reproducir animaciones (avance, retroceso y pausa) del funcionamiento de la máquina, p.ej., transiciones en un autómata finito durante el reconocimiento de una cadena.

**Interactivo** que permite explorar el comportamiento de la máquina especificando los datos de entrada, p.ej., reconocimiento de varias cadenas especificadas por el estudiante, para comprobar el funcionamiento en distintas situaciones.

**Generativo de primer orden** que permite a los estudiantes especificar la máquina que luego se visualizará, p.ej., el estudiante diseña el autómata y visualiza su comportamiento usando los niveles anteriores.

**Generativo de segundo orden** que permite visualizar el proceso de generación de la máquina, p.ej, el estudiante observa el proceso de construcción de la tabla de análisis SLR para una gramática que él ha especificado.

Sin embargo, las máquinas abstractas no dejan de ser algoritmos concretos, independientemente de su potencia para resolver problemas. Un trabajo de ámbito más general es la creación por parte de Naps et al. [158] de una taxonomía para diferenciar los distintos *niveles de implicación* de los alumnos con las visualizaciones de algoritmos. A continuación describimos brevemente cada uno de estos niveles:

**Sin visión** que caracteriza a aquellos entornos donde no se usa ninguna tecnología relacionada con la visualización.

**Visión** es el nivel más básico, donde el alumno ve la animación. Existen dos clases de visión: la *pasiva* donde el alumno no hace nada más, y la *activa* donde el alumno es capaz de controlar propiedades básicas como el sentido, el ritmo, la velocidad o seleccionar diferentes vistas.

**Respuesta** implica que el alumno, además de ver la animación, debe contestar a ciertas preguntas planteadas durante la misma.

**Cambio** permite al alumno aportar datos de entrada para el algoritmo, de forma que controla la ejecución del mismo.

**Construcción** requiere que el alumno genere la animación, ello implica que el alumno podría seleccionar qué partes de la animación se muestran y cuáles no. Esto se puede hacer de dos formas: codificando él mismo el algoritmo, aportando los datos de entrada y obteniendo los resultados en imágenes o animaciones; o utilizando una herramienta de dibujo que le permita generar dichas animaciones (incluso un generador de presentaciones valdría). Nótese por tanto, que en este nivel no se requiere la codificación del algoritmo.

**Presentación** requiere que se exponga una animación a una audiencia, para después obtener opiniones sobre ella o celebrar debates.

Podemos observar las relaciones existentes entre los niveles de exploración y los de implicación. El nivel de exploración estático se puede asimilar al nivel de implicación de visión. El interactivo se puede asociar al de cambio. Y finalmente, los dos niveles generativos se pueden asimilar al de construcción.

El orden de los niveles de la taxonomía propuesta por Naps et al. [158] supone un orden ascendente en cuanto a la implicación de los estudiantes con las visualizaciones. Y la idea subyacente es que "*cuanta más implicación con la animación, mejor es el aprendizaje*". Sin embargo, no hay estudios que respalden esta afirmación en toda su amplitud, es decir, no hay un cuerpo representativo de estudios que demuestren la existencia: ni de mejoras en el aprendizaje con respecto al primer nivel, ni el aumento del aprendizaje según se aumenta la implicación de los estudiantes. Lo que sí hay, son experiencias alentadoras que muestran los beneficios de los niveles de implicación en situaciones concretas, p.ej., uso de las animaciones en conjunción con otras características como evaluación automática, o con cierto tipo de estudiantes; o mejoras sólo en ciertos niveles de aprendizaje, p.ej., niveles en la taxonomía de Bloom [28]. A continuación mencionamos las más representativas.

En cuanto al nivel de respuesta, Grissom et al. [78] utilizaron un conjunto de animaciones de algoritmos publicadas en la Web. Probaron los tres primeros niveles de implicación. Aunque el aprendizaje aumentaba a la par que el nivel de implicación, sólo detectaron diferencias significativas entre los estudiantes que no usaron animaciones y aquellos que debían responder a preguntas mientras usaban las animaciones. Dichos resultados se midieron en el marco de la taxonomía de Bloom en los dos primeros niveles: conocimiento y comprensión. En este mismo nivel de implicación, Laakso et al. [118] realizaron

estudios comparando dos cursos consecutivos. El primero con una metodología clásica de clases magistrales y ejercicios escritos, y el segundo utilizando ejercicios de simulación. Dichas tareas consistían en la simulación del comportamiento de los algoritmos junto con la evaluación automática de las soluciones aportadas por los alumnos. Pudieron comprobar que el número de aprobados fue significativamente mayor en el curso en que se usaron las animaciones. En un ámbito más centrado en tutores basados en preguntas al estudiante, Kumar [117] detectó mejoras significativas si las evaluaciones de esas preguntas se acompañaban de visualizaciones, y más aún si a estas se le añadía explicaciones textuales.

Relacionados con el nivel de cambio, Lawrence et al. [124] comprobaron que existían mejoras significativas si se comparaba con el nivel que no usa visualizaciones, pero no con el nivel de visión. El ámbito de este estudio fue un algoritmo concreto, el algoritmo de Kruskal para el árbol de recubrimiento mínimo. Por otro lado, Ben-Bassat et al. [22] experimentaron con el aprendizaje de las características básicas del lenguaje Java. Clasificando a los estudiantes en tres clases (buenos, medios y malos), los resultados de la evaluación mostraron que los estudiantes medios obtuvieron mejores calificaciones. Los estudiantes buenos no necesitaban las animaciones, y los malos encontraban las tareas demasiado complejas para ellos. Moskal et al. [141] consiguieron resultados positivos en el ámbito de la orientación a objetos con alumnos que ellos califican de "riesgo". Este tipo de alumnos tiene poca experiencia en programación o pocos conocimientos matemáticos, y alta probabilidad de abandonar el estudio de la materia. Por último, Hansen et al. [83] comprobaron que visualizaciones con documentación hipertexto añadida y preguntas durante la reproducción de las animaciones, mejoraban los conocimientos aprendidos si se incluían actividades del nivel de cambio y mayor detalle en la información sobre el funcionamiento del algoritmo, ya que las visualizaciones con las que se comparaba, o eran analogías del algoritmo o animaciones centradas en mostrar el comportamiento del algoritmo con grandes cantidades de datos.

Stasko [213] introdujo ejercicios específicos de construcción de animaciones en su metodología docente. De hecho, la evaluación de los ejercicios se hacía en función de la corrección del algoritmo programado y la representatividad de la animación generada. Este experimento careció de grupo control, por lo que no pudo medir mejoras de aprendizaje. Sin embargo, detectó que los estudiantes eran más proclives a estudiar aquellos algoritmos para los que habían construido animaciones, y así lo comprobó en sus calificaciones.

Los estudios relacionados con el nivel de presentación también incluyen actividades de construcción de animaciones. Hundhausen [91] realizó un estudio etnográfico sobre la construcción y presentación de animaciones. Aunque no midió en términos cuantitativos el aprendizaje, sí obtuvo resultados cualitativos interesantes. Por ejemplo, el uso de herramientas típicas de animación de algoritmos, como las que usaba Stasko, distraía a los alumnos de las cuestiones importantes a la hora de diseñar las animaciones. Sin embargo, si se usaban métodos menos técnicos, como la confección de viñetas (o comics), su motivación mejoraba claramente, aumentando su participación y con ello las posibilidades de aprender más. Por otro lado, Hübscher-Younger y Narayanan [90] estudian los efectos de crear animaciones en un entorno donde los alumnos además debían debatir sobre las animaciones presentadas por otros, así como evaluarlas. Los estudiantes podían construir las animaciones según sus preferencias, usando tecnologías propias de visualización, presentaciones estilo PowerPoint, o incluso a mano. El resultado principal es que la construcción ayuda significativamente al aprendizaje de los algoritmos.

### 2.4.2. Revisión de sistemas

Desde el nacimiento de esta disciplina en los 80, son innumerables los sistemas que se han desarrollado para la investigación. Algunas compilaciones disponibles son [60, 63, 139, 215]. Sin ánimo de ser exhaustivos, mencionaremos los más significativos que han pasado por algún proceso de evaluación, identificando el grado de implicación que requieren de los estudiantes y los resultados que han obtenido en sus evaluaciones[10].

El proceso de generación visualizaciones de programas y algoritmos se divide básicamente en tres pasos: la producción de la información necesaria a visualizar, la generación de la representación gráfica de las visualizaciones, y su uso por parte de los estudiantes.

Siguiendo esta división podemos clasificar los sistemas de visualización de programas y algoritmos en tres clases: aquellos especializados en la generación de la representación gráfica de las visualizaciones, permitan o no la visión de estas, también llamados *motores de visualización*; aquellos especializados en la interacción de los alumnos, que ofrecen una interfaz para su reproducción permitiendo manipular el ritmo de la animación, la contestación a preguntas o la inserción de nuevos datos de entrada a los algoritmos, los llamaremos *interfaces de visualización*; y finalmente aquellos que acaparan todas las fases, que llamaremos *entornos de visualización*.

Debemos hacer dos consideraciones. Por un lado, no hemos tenido en cuenta la generación de la información a visualizar porque: o es un API que genera información codificada en ciertos lenguajes que luego los motores de visualización interpretarán; o está desarrollada ad-hoc, obviamente, en los entornos de visualización. Por otro, la diferenciación entre motores y entornos de visualización a veces no está tan clara, ya que normalmente todo motor de visualización tiene una utilidad para que los estudiantes interactúen con las visualizaciones que genera. Los diferenciaremos por la entidad que muestran y sus propios autores les atribuyen. Por ejemplo, un motor de visualización debe ser un desarrollo que ha sido descrito como tal y que permite su integración con otras interfaces distintas, ya sea porque existe documentación al respecto o porque realmente se ha hecho dicha integración.

#### Motores de visualización

El funcionamiento básico de un motor de visualización es recibir la especificación de la animación –un script– y generar su representación gráfica. La generación del script se suele hacer mediante la inserción de llamadas al sistema de visualización, dentro del programa fuente a visualizar. Cuando se ejecuta el programa, como efecto lateral se genera el script. La representación gráfica se facilitará a través de una interfaz propia o externa.

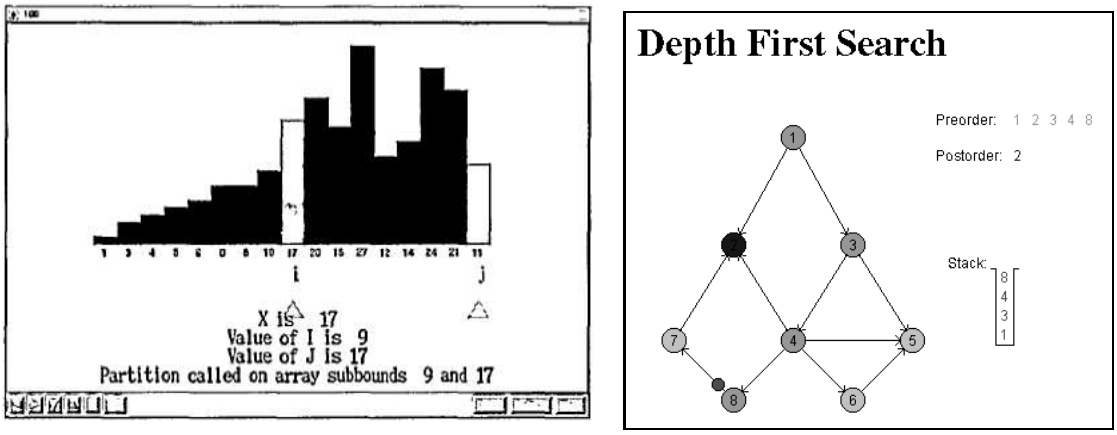
Algunos ejemplos de motores de visualización con interfaz propia son *JSamba*, *Jawaa*, *LJV* o Animal. Los clasificamos como motores, ya que esa es su aportación principal.

*JSamba*[11] (véase la figura 2.12a), es un motor de animación de algoritmos, recodificación en Java del sistema *Samba*, que a su vez es la evolución del sistema de animación de algoritmos XTango [212]. En el apartado anterior hemos mencionado los resultados obtenidos con esta familia de sistemas desde el punto de vista del nivel de implicación, y ahora los resumimos. La visión de animaciones con XTango no aporta mejoras pedagógicas [39]. Sin embargo, su uso en los niveles de implicación de cambio [124] y construcción –con JSamba– [213, 110] ofrecieron resultados positivos. En concreto, en el cambio detectaron mejoras cuantitativas en el aprendizaje, mientras que en la construcción sólo pudieron

[10] Algunos ya mencionados en el apartado anterior.
[11] http://www.cc.gatech.edu/gvu/softviz/algoanim/samba.html

detectar mejoras en la actitud de los estudiantes.

*Jawaa*[12] [174] (véase la figura 2.12b), permite ver las animaciones vía Web. Los scripts generados por los programas se guardan en un fichero de especificación (`.anim`). Además se genera una página Web con el mismo nombre que el fichero de la especificación, la interfaz asociada a Jawaa se ejecuta automáticamente cuando un navegador accede a esta página, reproduciendo la animación. Jawaa sólo se ha evaluado de forma cualitativa [2]. Los estudiantes se encontraron bastante incómodos teniendo que codificar las órdenes que generaban las animaciones, pero por lo menos la mitad opinaba que las animaciones les ayudaba. El último desarrollo es una utilidad para generar tanto objetos gráficos como animaciones más complejas sin teclear su código; aquellos se pueden utilizar luego en otra animación.




a) JSamba. [213] ©1997 ACM b) Jawaa. [2] ©2003 ACM

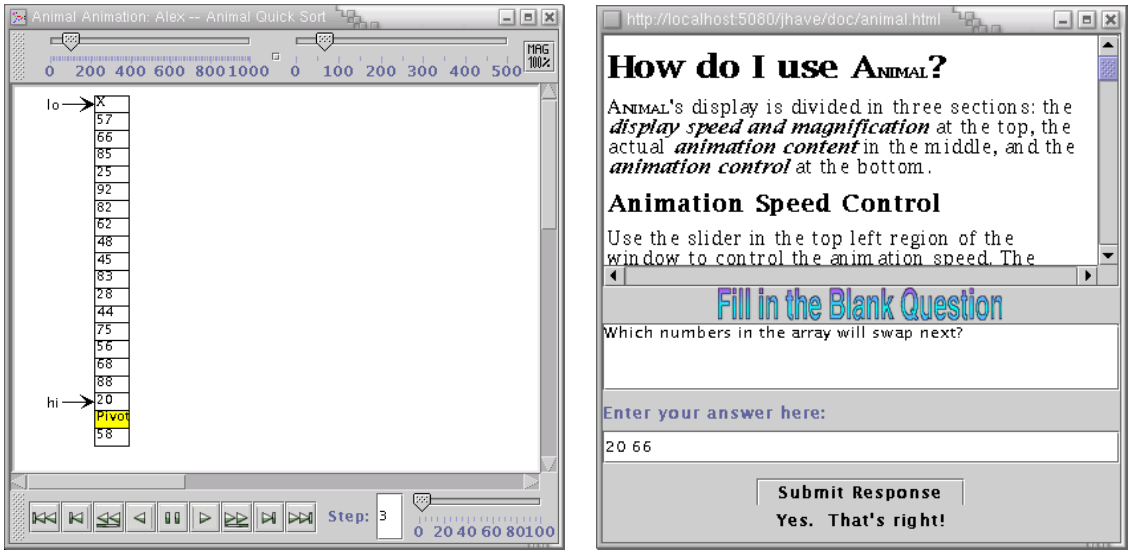
Figura 2.12: Animaciones con JSamba y Jawaa. (a) animación del algoritmo de ordenación QuickSort con JSamba, (b) animación del algoritmo de búsqueda en profundidad con Jawaa

*LJV*[13] [80, 81] genera visualizaciones de estructuras de datos en Java. Utiliza la información de tipo de los objetos en tiempo de ejecución; de esta forma es capaz de identificar automáticamente la forma de representarlos. El estudiante sólo tiene hacer una llamada al objeto para que se genere su visualización, nótese que esta es una solución mucho más simple que las anteriores. La presentación de las visualizaciones se hace con una interfaz simple, y las animaciones se componen de cada visualización generada durante la ejecución del programa. El uso de LJV se ha evaluado informalmente en el nivel de cambio. Los resultados son positivos en cuanto a la opinión de los estudiantes, pero no se han detectado mejoras en cuanto al aprendizaje.

ANIMAL[14] [196] (véase figura 2.13), implica de nuevo una notación más compleja, pero que le da más potencia, ya que llega incluso a ser capaz de generar preguntas durante la animación. ANIMAL ha sido evaluado –como motor de visualización– por expertos [193], pero no se ha evaluado su uso educativo con estudiantes.

Finalmente, tenemos dos motores de visualización que trabajan con interfaces externas, por lo tanto, aquí sólo los mencionamos. *Matrix*[15] [116] es un motor que se integra con las interfaces *MatrixPro* y *Trakla2*. Asimismo, *GAIGS* [153] trabaja con la interfaz *JHAVÉ*. Su última versión, *GAIGS XML*, tiene la posibilidad de especificar preguntas de distintos tipos –verdadero/falso, respuesta múltiple–,

[12] http://www.cs.duke.edu/csed/jawaa2/
[13] Lightweight Visualizer for Java, http://www.cs.auckland.ac.nz/~j-hamer/LJV.html
[14] http://www.animal.ahrgr.de/index.php3
[15] http://www.cs.hut.fi/Research/Matrix/




a) Ejemplo de animación. b) Pregunta durante la animación.

Figura 2.13: Animaciones y preguntas con ANIMAL. [193] ©2002 ACM

y su lenguaje se basa en XML, siguiendo las recomendaciones del grupo de trabajo sobre animaciones de algoritmos [159].

### Interfaces de visualización

Hemos identificado tres interfaces de visualización: *JHAVÉ*, *MatrixPro* y *Trakla2*. *JHAVÉ*[16] [154] se diseñó para trabajar con GAIGS, pero su última versión es capaz de integrarse además con Animal [157] y JSamba. Esta interfaz permite integrar las animaciones con explicaciones y preguntas durante su reproducción, además permite manejar bibliotecas de animaciones. Las evaluaciones realizadas con JHAVÉ se centraron en los tres primeros niveles de implicación. Los estudiantes aprendieron más –en los niveles conocimiento y comprensión de la taxonomía de Bloom– si usaban animaciones con preguntas que si no usaban animaciones [78].

*MatrixPro*[17] [106] es una interfaz para el motor de visualización Matrix. Esta interfaz está diseñada para que los profesores puedan utilizar las animaciones durante las clases.

*Trakla2*[18] [132] (véase figura 2.15), es la interfaz desarrollada para que los estudiantes trabajen con las animaciones generadas con Matrix. La interacción básica de los estudiantes es la visión de las animaciones, junto con la simulación del comportamiento de los algoritmos manipulando directamente las estructuras de datos. Su potencia se debe a los materiales y características que han añadido al uso de las animaciones, sobre todo la evaluación automática de las simulaciones de los alumnos y las explicaciones. La evaluación de Trakla [118] se realizó durante dos cursos consecutivos, el primero con un enfoque típico, y el segundo con el uso de Trakla2 en los niveles de visión, respuesta y cambio. Como resultado, detectaron que el porcentaje de aprobados fue significativamente mayor el segundo año.

Tratamos al sistema *HalVis* (*Hypermedia ALgorithm Visualizations*) [83] como una interfaz puesto su aportación principal son los materiales que las acompañan y la forma de utilizarlas. Como su propio

[16] http://jhave.org/
[17] http://www.cs.hut.fi/Research/MatrixPro/
[18] http://www.cs.hut.fi/Research/TRAKLA2/

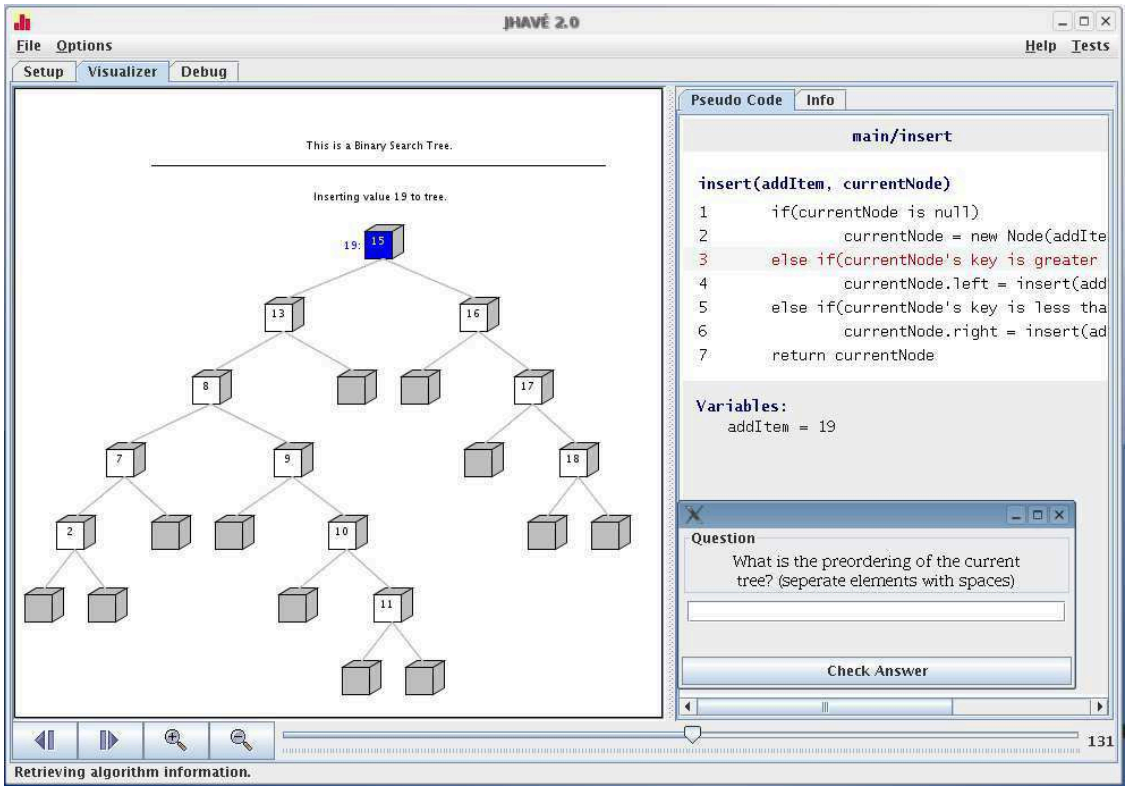



Figura 2.14: Ejemplo de animación con JHAVÉ. Cortesía de Thomas L. Naps [157]

nombre indica, HalVis acompaña las animaciones con explicaciones hipertexto; sin embargo su estructura es mucho más compleja y potente. Cada algoritmo documentado con HalVis se compone de tres animaciones a distintos niveles de complejidad: una animación general que los autores describen como una analogía acompañada de explicaciones, una animación detallada que se centra en las diferentes partes de ejecución del algoritmo, y una animación a gran escala que muestra el comportamiento de algoritmo con volúmenes de datos de entrada grandes. Cada animación está adaptada a su nivel detalle. Hemos de destacar que la segunda animación permite al estudiante probar el funcionamiento del algoritmo con sus propios datos de entrada, además de explorar diferentes niveles de detalle, desde sentencias concretas hasta partes diferenciadas del algoritmo (evidentemente, esto es dependiente de cada algoritmo). Para evaluar este sistema, hicieron varios estudios. Todas las comparaciones con libros de texto sobre algoritmos, clases magistrales típicas, o con explicaciones escritas por un profesor experto junto con animaciones generadas con Tango [212][19] dieron resultados significativos favorables a HalVis. La única comparación donde no se pudo demostrar que con HalVis se aprendía más, fue con una compilación de las mejores explicaciones, ejemplos y ejercicios de libros sobre algoritmos.

Debido a la diversidad de sistemas existentes en estas dos clases, interfaces y motores de visualización, y la eficacia pedagógica de sus enfoques, su integración empieza a ser un campo de investigación de creciente interés [105].

[19]En el apartado de motores de visualización hemos mencionado a su versión para el sistema Xwindows, XTango.

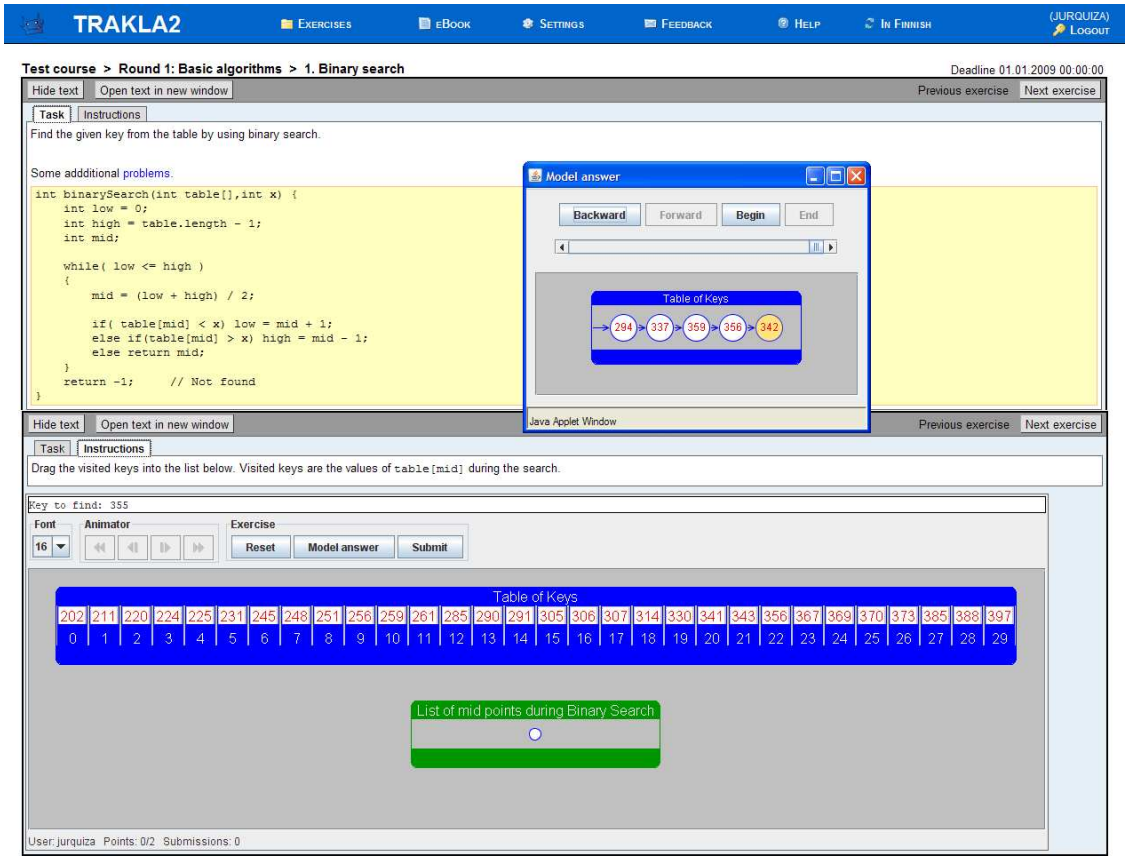



Figura 2.15: Animaciones y simulaciones con Trakla2

### Entornos de visualización

Los entornos de visualización son los más numerosos, habiendo identificado diez sistemas, que como dijimos anteriormente integran motor e interfaz de visualización. Los clasificaremos según las visualizaciones que producen:

**Entornos de visualización de algoritmos** que producen visualizaciones de alto nivel, normalmente de estructuras de datos abstractas.

**Entornos de visualización de programas** que producen visualizaciones de bajo nivel, detallando aspectos de los programas como variables, ámbitos de visibilidad, o evaluación de expresiones.

Dentro de los *entornos de visualización de algoritmos* encontramos dos sistemas independientes del lenguaje utilizado para programar los algoritmos que visualizan. Estos sistemas son *Swan* y *Alvie*.

*Swan*[20] [201], tiene claramente diferenciados el motor y la interfaz de visualización, cada uno de ellos con características interesantes. El motor de visualización utiliza la librería SAIL para la generación de las visualizaciones. Los autores pusieron especial atención a su facilidad de uso; p.ej., integra algoritmos para el dibujo de arrays, listas y grafos. La interfaz, SVI, permite el control de la reproducción de las animaciones con explicaciones textuales añadidas, y además permite interactuar con ellas de forma que podrían cambiar el comportamiento del algoritmo, p.ej, insertando claves en un árbol de búsqueda. Las

[20] http://research.cs.vt.edu/algoviz/Swan/

evaluación de Swan se restringe a una observación sobre la construcción de animaciones. Observaron que la notación es fácil de aprender y las animaciones fáciles de construir.

*Alvie*[21] [56] (véase figura 2.16), tiene un motor de visualización basado en XML. La generación se hace del mismo modo que en la mayoría de los motores, insertando comandos en el algoritmo a visualizar, que tras su ejecución habrán producido el script de la animación. Posteriormente, este script es interpretado por Alvie y mostrado con su interfaz. Por defecto, Alvie es capaz de mostrar animaciones acompañadas de explicaciones textuales, pero lo han utilizado en todos los niveles excepto el nivel de respuesta. Han hecho una evaluación cualitativa de su uso a nivel de visión, construcción y presentación; y los estudiantes encuestados estaban bastante satisfechos con la herramienta y el material provisto, 65 animaciones de distintos algoritmos tratados durante el curso.

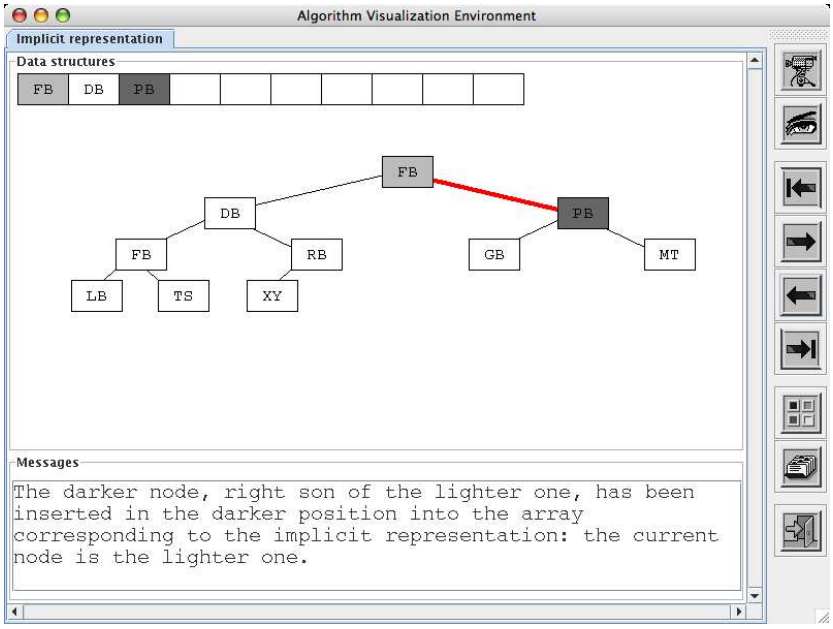



Figura 2.16: Ejemplo de animación con Alvie. [56] ©2007 ACM

El resto de entornos de visualización de algoritmos están fuertemente relacionados con el lenguaje que utilizan para programar los algoritmos que visualizan.

*MAVIS*[22] [114] es un sistema diseñado para entornos colaborativos remotos. Está dividido en un servidor que envía las visualizaciones a múltiples sistemas clientes. Carece claramente de un motor de visualización con capacidades gráficas, simplemente porque la representación gráfica de los objetos debe ser desarrollada por el propio usuario, profesor o estudiante. Tanto los algoritmos como las representaciones gráficas deben desarrollarse en Java. Sin embargo, proporciona una librería con mecanismos de control sobre las representaciones que se generan y muestran. Permite definir diferentes niveles de abstracción, manipulables por el profesor y los alumnos, así como diferentes niveles de complejidad, que cambiará según la carga de la red o la capacidad de los clientes. Han evaluado cualitativamente la herramienta centrándose en los niveles de visión, cambio y construcción; e ignorando el factor colaborativo/remoto. Los estudiantes valoraron positivamente los grados de abstracción, ya que les permitía explorar el comportamiento de los algoritmos a diferentes niveles de detalle, pero también porque les permitía hacer un diseño descendente de las animaciones desde los aspectos generales a los más

[21] http://www.algoritmica.org
[22] http://www.ee.technion.ac.il/∼ayellet/MAVIS-movies/

detallados.

*LEONARDO*[23] [55] (véase la figura 2.17), es un entorno integrado de programación para C, al que se le han añadido las facilidades de un entorno de visualización. Las visualizaciones se especifican mediante un lenguaje lógico [58], y su generación se produce mediante comentarios multilínea en cualquier parte del programa. El entorno ejecuta los programas en una máquina virtual propia y puede cambiar el sentido de la ejecución, y por lo tanto de la visualización. Han hecho evaluaciones de observación en los niveles de visión y construcción, obteniendo buena opinión por parte de los estudiantes. El último desarrollo es una interfaz para construir animaciones Web [29], pero no se ha evaluado.

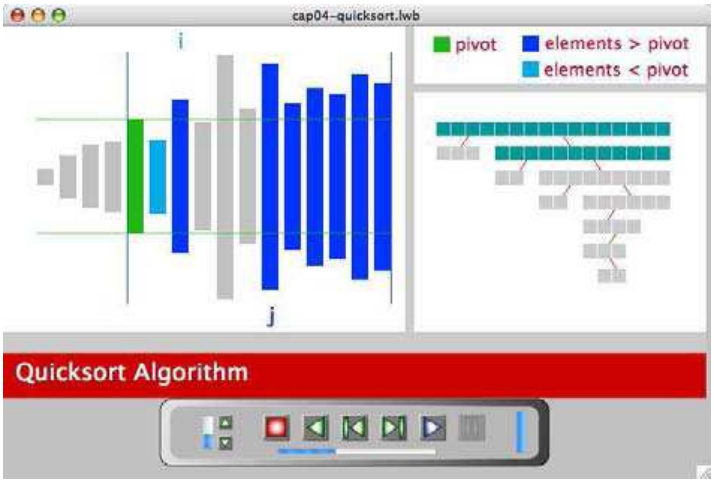



Figura 2.17: Ejemplo de animación con LEONARDO. [29] ©2006 ACM

*Alice*[24] [141] (véase la figura 2.18) se centra en la enseñanza de la orientación a objetos a través de un mundo virtual 3D manipulado con un pseudo-código propio. Los objetos de dicho mundo actúan según los algoritmos diseñados por el usuario. Se centran fundamentalmente en el nivel de cambio. Tras probarlo en el segundo curso de titulaciones de informática, han conseguido resultados cuantitativos positivos con aquellos alumnos que ellos califican de *riesgo*. Este tipo de alumnos tiene poca experiencia en programación o pocos conocimientos matemáticos, y alta probabilidad de abandonar el estudio de la materia.

*Alvis Live!*[25] [93] (véase la figura 2.19), permite editar programas en un pseudo-código propio, llamado SALSA. Estos programas se visualizan inmediatamente, en tiempo de edición. Así, cada vez que se edita alguna parte del algoritmo, se puede observar su comportamiento inmediato. También permite arrastrar objetos gráficos que luego se traducen en código en el algoritmo. Se ha evaluado mediante estudios de observación, obteniendo opiniones favorables de los estudiantes que construyeron animaciones con este sistema.

El último entorno de visualización de algoritmos que hemos identificado puede difuminar un poco la frontera, ya que es capaz de visualizar detalles de bajo nivel, pero su utilización se ha centrado en la programación de algoritmos. *jGRASP*[26] [86] (véase la figura 2.20), es un entorno de programación en Java con facilidades de visualización de algoritmos. Permite tener múltiples vistas con el código fuente, datos de bajo nivel de los objetos en tiempo de ejecución, así como visualizaciones de alto nivel. Estas

[23] http://www.dis.uniroma1.it/∼demetres/Leonardo/
[24] http://www.alice.org/
[25] http://eecs.wsu.edu/∼veupl/soft/alvis/index.htm
[26] http://www.jgrasp.org

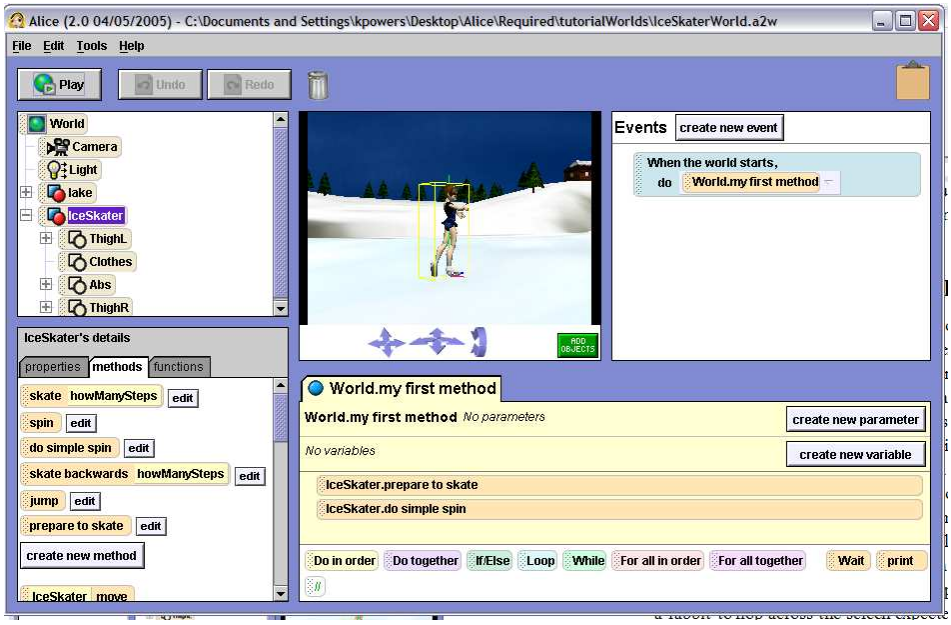



Figura 2.18: El entorno de visualización de algoritmos Alice. [179] ©2007 ACM

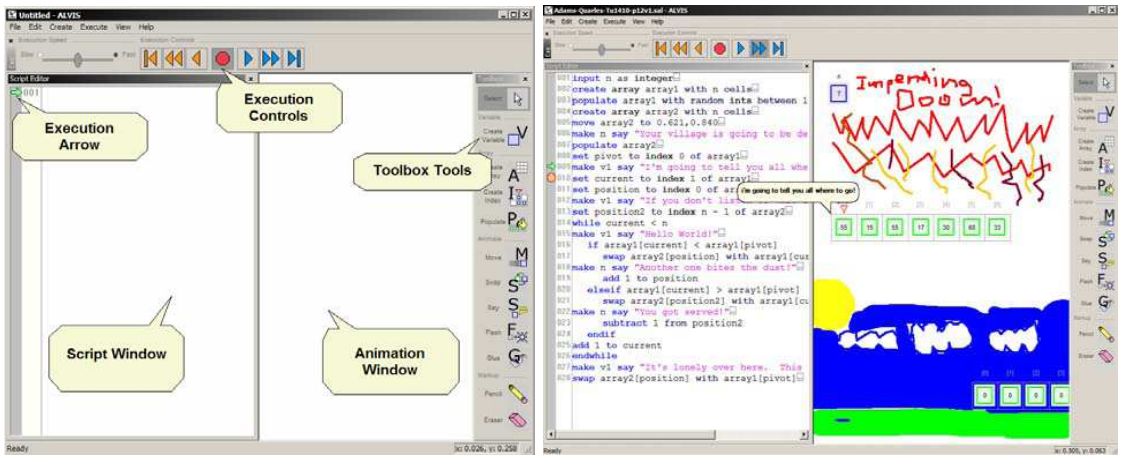



Figura 2.19: El entorno de visualización de algoritmos Alvis Live!. [92] ©2005 ACM

visualizaciones se han de desarrollar por los profesores, pero proporcionan una plantilla de partida que permite trabajar con su depurador (que proporciona al sistema la información durante la ejecución), su librería gráfica, y la interfaz de ventanas con barra de desplazamiento y solapas. Las visualizaciones de alto nivel son visualizadores externos dinámicos [57], pudiendo desarrollar varios para un mismo tipo de objetos. Una vez implementados los visualizadores externos, el funcionamiento del programa en ejecución produce las visualizaciones. Se centran en los niveles de visión y cambio, siempre dentro de su entorno con y sin los visualizadores externos. Sus resultados muestran que los estudiantes que usaron los visualizadores codificaron los algoritmos más rápido y de forma más correcta, y corrigieron algoritmos erróneos mejor y en menos tiempo.

Finalmente, hemos identificado tres *entornos de visualización de programas*: *Jeliot 3*, *VIP* y *Teaching Machine*. *Jeliot 3*[27] [140] (véase la figura 2.21) se centra en la visualización de programas codificados

[27] http://cs.joensuu.fi/~jeliot/

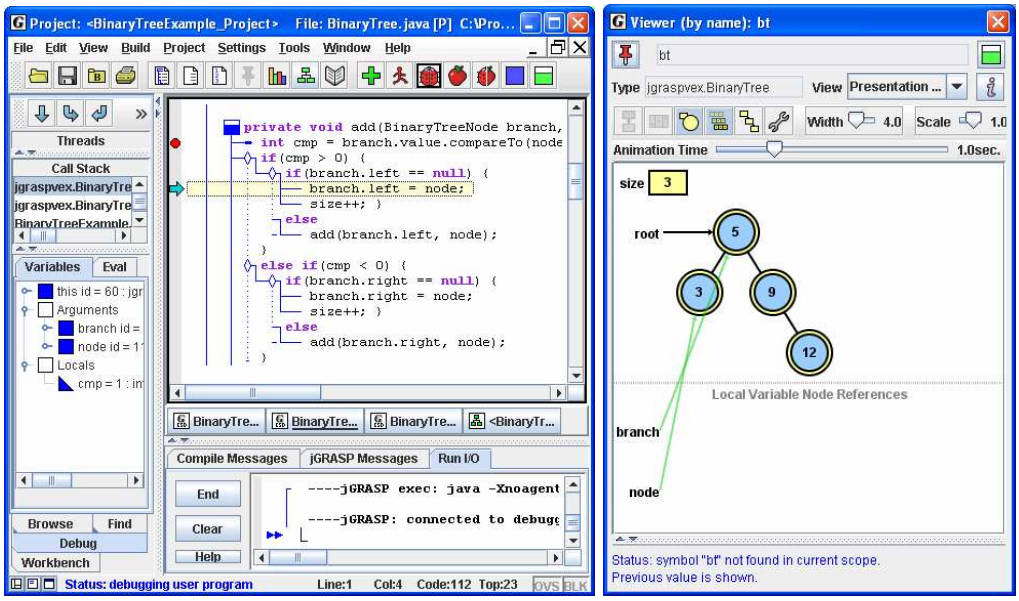



Figura 2.20: El entorno de visualización de algoritmos jGRASP. [57] ©2007 ACM

en Java. Su interfaz permite editar programas en Java y visualizar inmediatamente su ejecución. Está pensado para asignaturas de introducción a la programación, por lo que soportan las características básicas del lenguaje Java, pero no las de orientación objetos, p.ej., herencia. Su utilización se encuadra en los niveles de visión y cambio. La evaluación más significativa de este sistema, la hicieron con una versión anterior, *Jeliot 2000* [22]. Clasificaron a los estudiantes en tres clases (buenos, medios y malos). Los resultados de la evaluación muestran que los estudiantes medios obtuvieron mejores calificaciones con la herramienta que sin ella. Los estudiantes buenos no necesitaban de la herramienta, y los malos la encontraban demasiado compleja para ellos. Los últimos esfuerzos se han dedicado a su integración con *BlueJ*[28] y la generación automática de preguntas [144].

*VIP*[29] [240] es un entorno para un subconjunto del lenguaje C++ sin orientación a objetos. La ejecución de los programas es interpretada, pudiendo visualizar: el punto de ejecución actual, incluso a nivel de expresión; la evaluación en detalle de expresiones condicionales y aritméticas; las variables locales y globales, diferenciando gráficamente su utilización (lectura o escritura). Permiten dar marcha atrás, así como los controles típicos de ejecución paso a paso y control de la velocidad. También permiten producir explicaciones textuales asociadas a puntos concretos del código fuente. Dentro de ellas se pueden usar valores de variables, y pueden incluso controlar su visualización, p.ej., la segunda vez que se ejecute la sentencia. Este tipo de explicaciones se especifica mediante comentarios por lo que no afectan a la compilación del programa en otro compilador. También es capaz de bloquear la edición de ciertas partes del código fuente. Así un estudiante podría editar sólo una función, mientras que el resto del código lo proporciona el profesor. La evaluación de VIP [1] se ha centrado en los niveles de visión, cambio y construcción. Las tareas consistían en ver visualizaciones de programas y cambiar su comportamiento en el editor. Han comprobado que el uso de la herramienta ha mejorado significativamente los resultados de los estudiantes, aunque no pueden decir si es porque mejora el aprendizaje o porque consigue que los alumnos trabajen durante más tiempo con los ejercicios. Los últimos esfuerzos se han dedicado a la ampliación del sistema para permitir código oculto, de forma

[28] http://www.bluej.org/
[29] http://www.cs.tut.fi/∼vip/en/

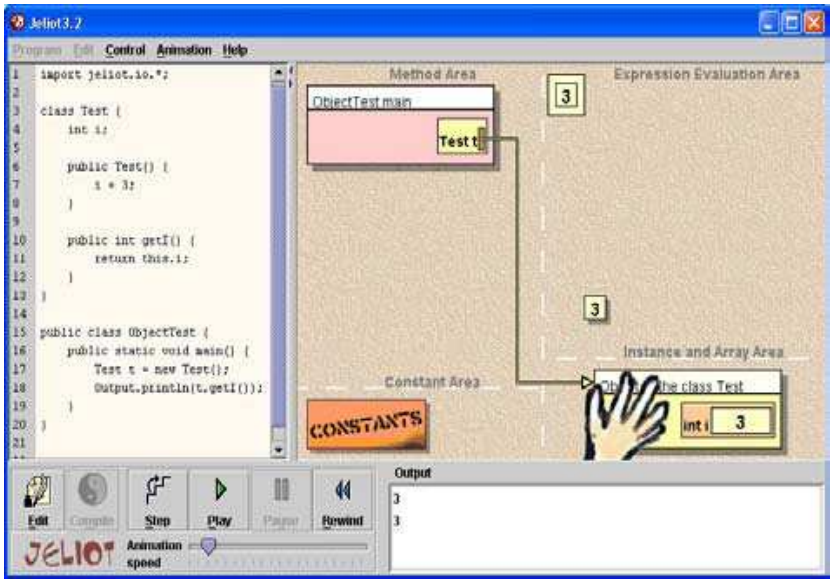



Figura 2.21: El entorno de visualización de programas Jeliot 3. [140] ©2004 ACM

que el profesor puede incluir pruebas que evalúen automáticamente las soluciones de los alumnos.

*Teaching Machine*[30] [165] es un sistema similar a jGRASP en lo que a información de depuración se refiere: código fuente, punto actual de ejecución, pila, evaluación de expresiones, consola, y además añade la visualización de la tabla de símbolos para las variables de los distintos ámbitos y una vista de la memoria dinámica con los contenidos de variables y punteros, estos últimos mediante arcos dirigidos. Han evaluado la herramienta en los niveles de visión y cambio, mediante observaciones cualitativas [37]. Los resultados de estas observaciones indican que los estudiantes están satisfechos con la herramienta.

#### Otros desarrollos interesantes

Como dijimos en un principio, los sistemas existentes son innumerables. Hemos encontrado otros sistemas interesantes sin evaluaciones de ningún tipo.

Así, podemos mencionar herramientas como *LIVE* [40], para la visualización interactiva de punteros y listas; *JVALL* [59], que visualiza listas enlazadas en Java; *SKA* [82], que se centra más en algoritmos de grafos; o el desarrollado por LaFollette et al. [119] que permitía visualizar estructuras de datos C/C++ compuestas por punteros y tipos básicos.

Por otro lado, hemos encontrado desarrollos recientes como *IMPROVE*[31] [130] y *WADEIn II*[32] [38], que representan desarrollos para incluir la tecnología adaptativa en las visualizaciones. Y el paquete de animaciones sobre árboles realizado por Rößling y Schneider [195], basado en el motor de visualización ANIMAL, que trabaja con diferentes niveles de implicación, genera preguntas, y proporciona explicaciones textuales adaptables al estudiante.

### 2.4.3. El papel de los profesores

En los apartados anteriores hemos presentado diferentes enfoques en el uso de las animaciones por parte de los estudiantes. Se trata de conseguir que el alumno se involucre de otra forma con la

[30] http://www.engr.mun.ca/∼mpbl/content/research/teachingMachine.htm
[31] http://www.sis.pitt.edu/∼paws/project_improve.htm
[32] http://www.sis.pitt.edu/∼paws/system_wadein.htm

animación, consiguiendo que la animación sea el vehículo del conocimiento que se quiere transmitir.

Sin embargo las animaciones no se han adoptado ampliamente como recurso pedagógico. Aunque, como hemos visto, existen experiencias positivas en cuanto a su efecto pedagógico, y son un recurso atractivo para estudiantes y profesores, el hecho es que no se usan tanto como en un principio podría parecer.

En el informe del grupo de trabajo sobre visualización de algoritmos del *ITiCSE'2003*[33], Naps et al. [155] estudian el porqué de esta situación. La idea general sería que, al igual que en marketing, no sólo se busca convencer a quien va a consumir el producto sino también a quien toma la decisión de comprarlo. Para que las animaciones se utilicen más ampliamente, debemos convencer al responsable de la decisión de incluir estas animaciones en la metodología pedagógica del curso en cuestión, el profesor. No basta con que ayuden a los alumnos y les resulten atractivas; hay que convencer al profesor de que vale la pena utilizaras en sus cursos.

Para la realización del informe, se hizo una encuesta a un conjunto de profesores, preguntándoles por qué no utilizan/utilizarían animaciones en sus cursos. El resultado se puede resumir en tres palabras: *"no hay tiempo"*. De las cinco razones más mencionadas por los profesores encuestados, cuatro eran relativas al tiempo. En concreto, al tiempo dedicado a: buscar buenos ejemplos para usar con las animaciones, aprender a manejar las herramientas de visualización, construir las visualizaciones y adaptarlas al enfoque pedagógico de los contenidos del curso. La quinta razón se refería a la ausencia de herramientas eficaces, pero esto ya lo hemos tratado en los apartados anteriores.

Es un hecho que las herramientas de animación construidas se centraban en el estudiante, ignorando completamente al otro usuario principal, el profesor. Para tratar de solventar los problemas mencionados anteriormente, Naps et al. [155] proponen una serie de recomendaciones para los futuros sistemas de animación de programas y algoritmos como: que sean independientes de la plataforma; más generales en cuanto los conceptos que permitan animar, con una interfaz común; adaptarlos a herramientas de enseñanza ya existentes, como los libros electrónicos, [128, 194]; dar flexibilidad para que el profesor pueda adaptar la herramienta a sus necesidades [177]; y otros aspectos secundarios a la herramienta pero muy importantes como la existencia de manuales, ayuda on-line, o un Web site con información actualizada. A estas características se pueden añadir otras propuestas por Pollack y Ben-Ari [177] como la existencia de visualizaciones ya hechas con relación muy cercana al curso y la utilidad de la herramienta para el profesor.

En los últimos años existe un creciente interés por conseguir que la creación de animaciones lleve poco o ningún esfuerzo al profesor. Así, Ihantola et al. [96] caracterizan el esfuerzo que se debe dedicar para la adopción de un sistema de animación de algoritmos desde tres puntos de vista: ámbito de aplicación en la materia impartida en el curso, la facilidad para adaptarse a los distintos escenarios de uso, y las distintas formas de interacción con las animaciones.

## 2.5. Resumen

Como hemos comentado, el uso correcto de las tecnologías de visualización ayuda a la formación de modelos mentales sobre conceptos abstractos. Y tanto la programación como el diseño de algoritmos son campos donde se trabaja con conceptos abstractos. Por esta razón, la visualización de programas y algoritmos son áreas donde existen gran cantidad de desarrollos y esfuerzos investigadores. Así,

[33] Innovation and Technology in Computer Science Education

podemos encontrar revistas internacionales de índice de impacto como *Journal of Visual Languages and Computing*[34] y congresos internacionales de especial relevancia como el *ACM Symposium on Software Visualization*[35], el *IEEE Symposium on Visual Languages and Human-Centric Computing*[36], o el *Program Visualization Workshop*[37].

Sin embargo, a pesar de todos estos esfuerzos, los beneficios educativos de las visualizaciones de programas y algoritmos no son inmediatos. No sólo hay que adecuar las características de las representaciones gráficas, sino que como Hundhausen et al. [94] comprobaron, lo realmente importante es el modo en que los estudiantes usan esas visualizaciones. Así, Naps et al. [158] desarrollaron la taxonomía de niveles de implicación, donde se han enmarcado numerosos estudios. Dichos estudios sólo han podido demostrar beneficios educativos de forma cuantitativa en los niveles de visión, respuesta y cambio. El nivel de cambio parece ser la frontera de estos beneficios educativos, y parte de las mejoras de aprendizaje detectadas en niveles de menor implicación se basan en añadidos significativos como la evaluación automática [118], o explicaciones textuales [83]. Mientras que el resto de estudios, tanto en la construcción como en la presentación, detectan mejoras únicamente en la actitud de los estudiantes.

Como ya hemos dicho en el capítulo de introducción, uno de los objetivos de esta tesis es demostrar que la construcción de animaciones que no requiere gran esfuerzo por parte del estudiante aporta beneficios educativos.

La gran mayoría de los esfuerzos en visualización del paradigma funcional son de carácter tecnológico o profesional. Los pocos que se pueden enmarcar en el ámbito educativo carecen de evaluaciones formales que demuestren sus beneficios pedagógicos. Además, algunos de ellos carecen de las características que hemos mencionado en apartados anteriores.

Así, *KIEL* visualiza el árbol de sintaxis abstracta de las expresiones funcionales. Aunque estos árboles sean el núcleo del código que interpreta la máquina virtual, no se corresponden con el modelo mental que el alumno debe formar sobre los programas funcionales. Utilizando una analogía muy cercana, ¿sería eficaz enseñar C++ visualizando el código objeto que se ejecuta en el procesador?

Por otro lado, para usar *Evaltrace* hay que tener conocimientos profundos de LaTeX. Además genera representaciones muy estáticas, más cercanas al modelo mental de un programador experto, y que sólo se podrían usar en el nivel de visión. *KAESTLE & FooScape* muestran un comportamiento más dinámico, pero el ámbito que cubren es muy pequeño (activación de funciones y listas enlazadas), además de ser muy antiguos, y hoy en día inutilizables.

Finalmente, *DrScheme* es un sistema mucho más completo: es capaz de trabajar con diferentes grados de complejidad del lenguaje, se puede usar en niveles de visión y cambio, y dispone de un Web site actualizado con material auxiliar como manuales y demostraciones. Sin embargo, carece de evaluación educativa y está fuertemente relacionado con el lenguaje de programación Scheme.

Nuestra propuesta se centra en la enseñanza de la programación funcional, y en un modelo del uso de las animaciones como recurso educativo de entidad propia. Por ello, pondremos especial énfasis en el diseño de sus contenidos así como en su proceso de construcción, potenciando los niveles de implicación de visión y construcción respectivamente. Según la clasificación anteriormente utilizada, nuestra propuesta es un *entorno de visualización*, ya que cubre aspectos de la propia generación de las animaciones, junto con su uso educativo. El modelo de animación resultante se ha implementado en un

[34] http://www.elsevier.com/locate/jvlc
[35] http://www.st.uni-trier.de/~diehl/softvis/org/softvis08/
[36] http://vlhcc08.cs.unibw.de/
[37] http://www.algoanim.net/pvw2006/

entorno de programación concreto, aunque no está ligado al lenguaje de programación. En el siguiente capítulo describiremos dicho entorno, y su ampliación para adaptar el modelo que proponemos.

# Capítulo 3

# Marco general de trabajo

Las aportaciones de esta tesis se agrupan en dos campos de investigación distintos, aunque necesariamente relacionados: la *interacción persona-ordenador*, y la *informática educativa*. Partimos de un entorno de programación funcional con facilidades para la generación de animaciones llamado *WinHIPE*. Las aportaciones de esta tesis han conseguido que tanto las animaciones generadas por dicho entorno, como el propio proceso de construcción sean herramientas educativas eficaces.

Para facilitar la comprensión de estas aportaciones, en este capítulo damos una visión global del marco de trabajo en el que se integran. Así, en primer lugar describimos el estado de desarrollo del entorno WinHIPE al comienzo de esta tesis, nuestro punto de partida. Y después explicamos las aportaciones realizadas en esta tesis, así como su ubicación en el entorno.

## 3.1. El punto de partida

WinHIPE es un entorno integrado de programación funcional con facilidades para la visualización estática y animación de programas. La animación de programas pretende mostrar ciertos aspectos de su ejecución, por lo tanto, en primer lugar describiremos el *modelo de ejecución* de programas funcionales implementado en WinHIPE. Una vez descrito este modelo, especificaremos el *modelo de visualización estática*, donde veremos, cómo y con qué características se generan las visualizaciones de cada paso de ejecución de los programas en WinHIPE. Finalmente, detallaremos el *modelo de animación*, donde veremos cómo construir y reproducir las animaciones. La figura 3.1 muestra cómo se integran estos tres modelos en WinHIPE.

En lo que a las animaciones respecta, el diseño de WinHIPE ha estado influenciado por lo requisitos propuestos por Böcker et al. [15], a saber: integración de las facilidades de animación en el entorno, y automatización de su construcción. Por otro lado se pretendió que tanto el uso como la construcción fueran lo más sencillos posible. Para todo ello, se dotó a WinHIPE de las siguientes características:

- Configuración sencilla de la representación gráfica asociada a las visualizaciones estáticas.
- Generación semiautomática de animaciones discretas, simplificando su proceso de construcción.
- Reproducción de las animaciones sencilla e *integrada* en el IDE.
- Reproducción de las animaciones sencilla e *independiente* del IDE.
- Posibilidades de integración de animaciones con explicaciones textuales.

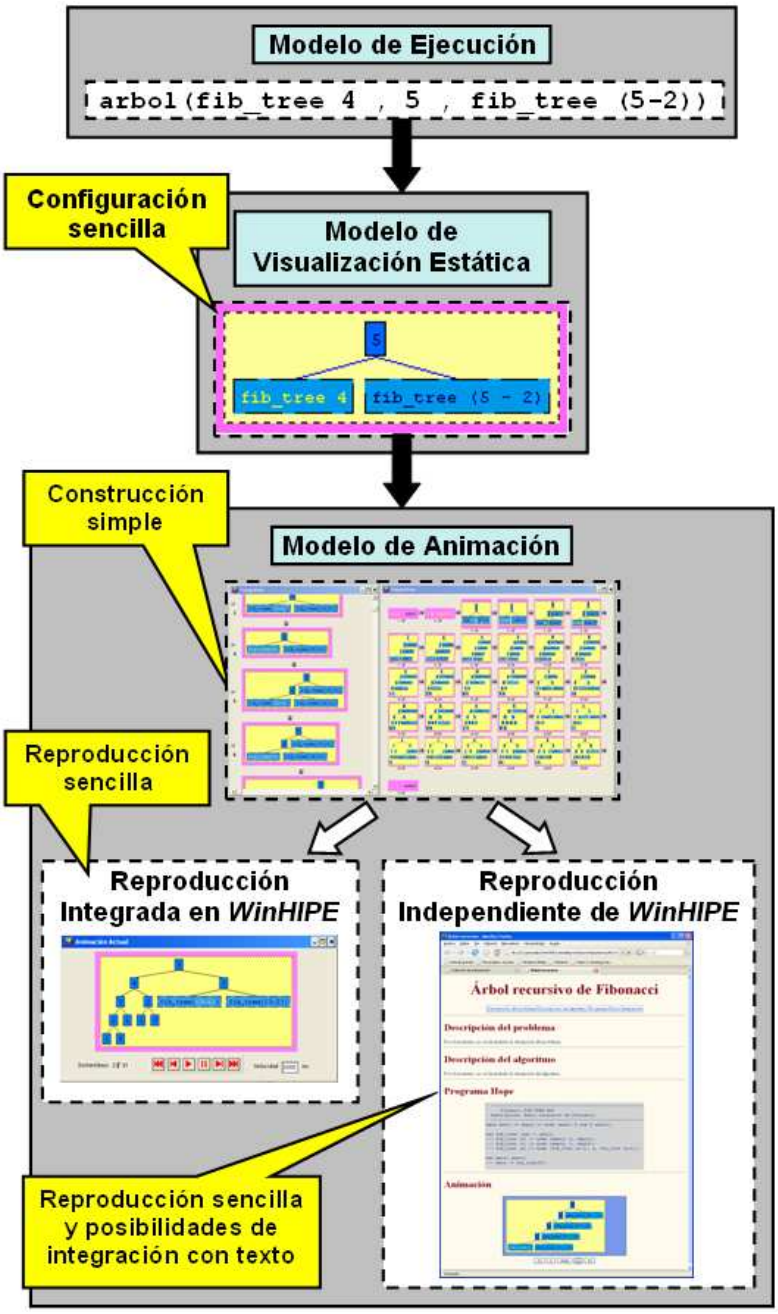



Figura 3.1: Integración de los modelos de ejecución, visualización estática y animación de programas funcionales en WinHIPE.

### 3.1.1. El modelo de ejecución de programas funcionales

Como ya explicamos anteriormente en el apartado 2.3.1, la ejecución de programas funcionales se puede entender como un proceso de reescritura. Durante este proceso, una expresión inicial se va modificando en cada paso de ejecución del programa, hasta llegar a una expresión final en forma normal, que es aquella sobre la que no es posible aplicar ninguna transformación más. Aunque en un principio se pensaba en todas las posibles estrategias de evaluación, en este modelo de ejecución sólo se tuvo en cuenta la estrategia de evaluación impaciente.

El proceso básico de ejecución de los programas funcionales es la reescritura de expresiones y subexpresiones, pero para mostrar la ejecución de un programa no es obligatorio mostrar la totalidad de los pasos de ejecución realizados. La ejecución paso a paso es la operación básica, pero sobre ella se pueden construir un conjunto de operaciones que permite mostrar solamente los pasos de ejecución en los que se esté interesado; por ejemplo, la llamada a determinadas funciones, o el estado de la ejecución en diferentes puntos del programa.

Velázquez-Iturbide [233] propuso un modelo de ejecución que permitía manipular la ejecución de programas funcionales de forma que se adaptara mejor a las necesidades del usuario. Dicho modelo se implementó en la aplicación HIPE (*Hope Integrated Programming Environment*), un entorno integrado de programación para el lenguaje funcional Hope+ [173]. Así, tomando como base el paso de reescritura simple, se proporcionaban las siguientes formas de controlar la ejecución:

**Secuencia de *n* pasos de reescritura** que consiste en ejecutar las $n$ transformaciones siguientes y mostrar la expresión en el estado resultante.

**Ejecutar hasta un punto de control** que consiste en ejecutar pasos de reescritura hasta que se alcanza un determinado lugar en el código fuente del programa, llamado punto de control. Por defecto, se ofrecen puntos de control en cada función definida por el usuario.

**Evaluar el redex** que consiste en ejecutar tantos pasos de reescritura como sean necesarios para transformar la subexpresión correspondiente al redex en su forma normal.

**Completar la evaluación** que consiste en ejecutar todos los pasos de reescritura restantes hasta obtener la forma normal de la expresión original.

En la figura 3.2 se muestra un ejemplo del control de la ejecución de un programa que calcula el factorial de un número. La expresión a evaluar es: `fact 4`. En cada estado de ejecución se destaca el redex correspondiente en color rojo, y cada operación de ejecución realizada se representa en color azul y con una flecha etiquetada: si no tiene etiqueta, se ha ejecutado un paso de reescritura; si la etiqueta es un número "*n*", se han ejecutado $n$ pasos de reescritura; si es la letra "`r`", se han ido ejecutando pasos de reescritura hasta encontrar un punto de control (en este caso la llamada a la función `fact`); si es la letra "`x`", se ha evaluado el redex; y si es el carácter "`*`", se han ejecutado todos los pasos de reescritura restantes hasta evaluar la expresión inicial.

Este modelo de ejecución es el que ha heredado WinHIPE. Además se le añadió la facilidad de *marcha atrás en la ejecución* [146]. De esta forma, un usuario podía volver a estados anteriores de ejecución y cambiar los pasos dados para ver otros estados de ejecución intermedios.

### 3.1.2. El modelo de visualización estática de programas funcionales

Todo modelo de visualización ha de tomar decisiones sobre la información que manipula y el proceso que sigue hasta obtener la visualización. Esto se traduce en responder a cinco preguntas:¿Qué información se visualiza? ¿Qué datos se recogen para la visualización? ¿Cuándo se recogen? ¿Cuándo se visualizan? y ¿Cuál será la representación visual de los datos?

Según este modelo, se visualizan tanto el código en ejecución como las estructuras de datos manipuladas por el código. El origen de esta información viene del modelo de ejecución. Así, para recoger los datos sobre un estado de ejecución en concreto se usa la expresión intermedia correspondiente. Como las expresiones se generan en tiempo de ejecución, tanto su recogida, como su visualización se harán

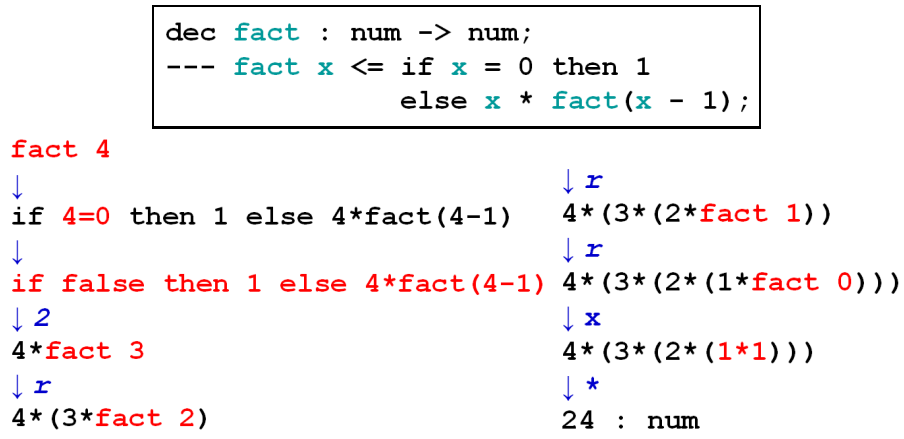



Figura 3.2: Secuencia de ejecución de la expresión `fact 4` utilizando diferentes pasos de ejecución. En el recuadro mostramos el código fuente de la función que calcula el factorial. Inmediatamente debajo, en azul se destacan los pasos de ejecución, y en rojo el redex de cada paso

en tiempo de ejecución. Todas estas decisiones son necesarias, pero las decisiones que más impacto van a tener sobre el resultado final son aquellas relacionadas con la representación visual de los datos.

La representación textual de las expresiones intermedias descritas hasta el momento es de por sí una forma de visualizar la ejecución de programas funcionales. De hecho, en los ejemplos contemplados hasta el momento, la representación textual ha servido para expresar el estado de ejecución del programa sin ningún problema. Como cualquier lenguaje de programación, los programas funcionales tienen un código fuente donde las diferentes estructuras léxicas y sintácticas definen su contenido. Este modelo de visualización estática permite asociar diferentes representaciones a las construcciones del código fuente utilizando técnicas de pretty-printing p.ej., en una sentencia `if-then-else` se dedica una línea al `if`, otra distinta al `then` y otra al `else`.

Sin embargo, para expresiones más complejas, utilizar únicamente la representación textual puede llegar a ser contraproducente. Las expresiones complejas pueden ser resultado tanto del uso avanzado de construcciones simples del lenguaje, p.ej. sentencias de bifurcación anidadas, como de la utilización de tipos de datos estructurados, p.ej. listas o árboles.

Jiménez-Peris et al. [102] describieron una extensión del entorno HIPE para generar representaciones mixtas de estas expresiones, tanto gráficas como textuales, permitiendo comprender mejor los pasos intermedios en la ejecución de los programas. Un ejemplo de estas representaciones se puede ver en la figura 3.3, donde se muestra la visualización de cuatro expresiones distintas sobre árboles. Nótese que la estructura arbórea se representa gráficamente, mientras que el contenido de los nodos se representa textualmente.

La posibilidad de utilizar este tipo de representaciones para tipos de datos específicos, se implementa enlazando los constructores de tipos con su representación gráfica. Así el tipo de datos árbol binario, que en Hope tendría la siguiente definición:

```
data arbol = Nodo( arbol # num # arbol) ++ NodoVacio;
```

está constituido por dos constructores: `Nodo` para aquellos nodos con algún contenido, y `NodoVacio` para aquellos nodos que no tienen ningún contenido. Ambos tienen su representación gráfica. Como se puede ver en la figura 3.3 un nodo no vacío se representa textualmente por su contenido (un número entero, en este caso), y dos líneas conectando la representación gráfica de sus dos nodos hijos; un nodo

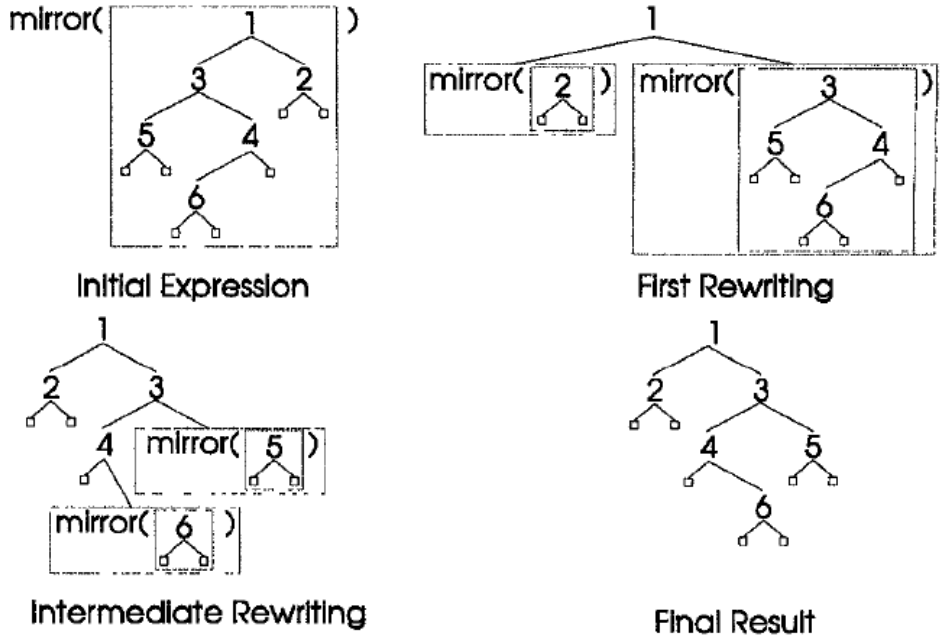



Figura 3.3: Representaciones gráficas de expresiones funcionales con árboles en HIPE. [102] ©ACM 1996

vacío se representa por un cuadrado pequeño. El modelo de visualización también define por defecto el aspecto visual de otros tipos, por ejemplo: los números enteros y las cadenas de caracteres tienen su representación textual natural, las tuplas se representan textualmente como una lista entre paréntesis de elementos separados por comas, y las listas de elementos tienen una representación gráfica como la presentada en la figura 3.4.

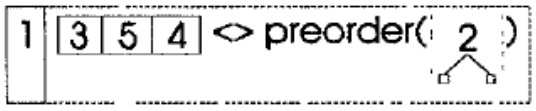



Figura 3.4: Representaciones gráficas de expresiones funcionales con listas en HIPE. Nótese cómo se pueden mezclar representaciones. En este caso, un elemento de la lista es una expresión de concatenación de una lista de números con el resultado de procesar un árbol. [102] ©ACM 1996

El modelo de visualización estática también contempla la posibilidad de adaptar la representación de las expresiones a las necesidades o preferencias del usuario. Dicha adaptación se implementa tanto a nivel de contenido como de formato.

Por defecto, se muestra todo el contenido de las expresiones intermedias. Sin embargo, en muchas ocasiones no es necesario mostrar todo este contenido sino la parte que se va a evaluar a continuación. Velázquez-Iturbide [234] describe la adaptación de la técnica de *ojo de pez* [69, 70] a la simplificación de expresiones funcionales. Gracias a esta técnica el usuario puede elegir la cantidad de información visible al rededor del *redex* manipulando un umbral de visibilidad, por encima del cual nada es visible. La figura 4.5 muestra la expresión `if 4=0 then 1 else 4*fact(4-1)` con cuatro umbrales de visibilidad distintos.

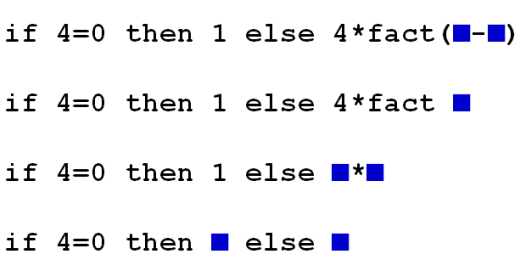



Figura 3.5: Control del contenido mostrado de una expresión funcional. Los cuadrados azules representan contenidos que están por encima del umbral de visibilidad, y por lo tanto, se ocultan. La primera expresión se corresponde con un umbral de valor 5, la siguiente con 4, 3 y finalmente 2

Además, este modelo permite dar información sobre el flujo del control de ejecución en el programa. Este es gobernado por la estrategia de ejecución –en nuestro caso impaciente–, las sentencias de bifurcación, invocación de funciones y la selección de las cláusulas a ejecutar en cada momento. Desde el punto de vista estático se puede usar el sangrado, ya que casi todos los puntos antes descritos tienen su representación sintáctica, permitiendo el uso de técnicas de pretty-printing. Pero desde un punto de vista más dinámico, resaltando el redex de la expresión se destaca la siguiente parte del código a ejecutar.

La personalización de la representación de las expresiones a nivel de formato permite al usuario controlar los parámetros que fijan la apariencia visual de estas. Existen gran cantidad de parámetros, por lo que se debe proveer al usuario de una interfaz que permita manipular estos fácilmente. Estas mejoras se integraron en una nueva versión del entorno, basada en Windows, llamada *WinHIPE*. De esta nueva versión, Velázquez-Iturbide & Presa-Vázquez [239] destacan tanto las mejoras del entorno de programación, p.ej., la gestión de proyectos, como las posibilidades de generación y personalización del formato de las visualizaciones. Así, las visualizaciones estáticas se pueden generar fácilmente a base de opciones de menú, permitiendo además controlar parámetros como el color de los nodos de un árbol, representación gráfica de la lista vacía, o el color del redex, entre otros. La figura 3.6 muestra una pantalla de configuración para parte de estos parámetros.

En resumen, el modelo de visualización estática permite obtener representaciones mixtas de texto y gráficos donde el usuario es capaz de configurar, a través de una interfaz gráfico, tanto su contenido como su apariencia visual.

### 3.1.3. El modelo de animación de programas funcionales

En este caso, la información a visualizar es el comportamiento del programa durante su ejecución. Los datos que se utilizan para construir la animación son las visualizaciones estáticas. Así, la animación de un programa funcional será la secuencia de visualizaciones correspondiente a las expresiones por las que ha pasado su ejecución. Por lo tanto, los datos se recogen en tiempo de ejecución, pero la animación no se mostrará hasta tener todas las visualizaciones. Según esto, la animación no se mostrará hasta que la ejecución del programa no haya terminado, por lo tanto este modelo genera animaciones *post-mortem*.

Además, el modelo de animación va a contemplar el doble objetivo de la sencillez y automatización de los procesos relacionados con las animaciones. Para ello, Naharro-Berrocal et al. [150] utilizan la

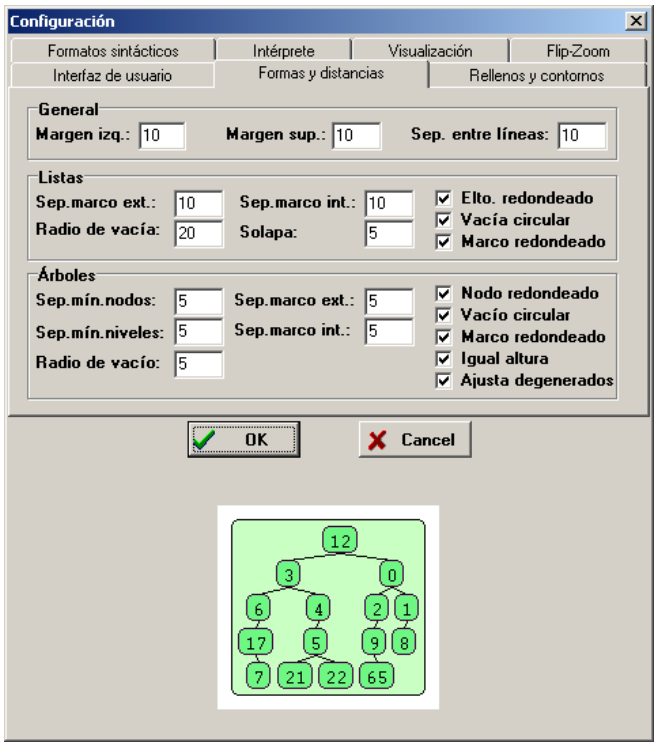



Figura 3.6: Ventana de configuración de los aspectos gráficos de formas y distancias relativos a árboles y listas

metáfora del documento ofimático. Así, las animaciones son documentos que se podrán crear, usar, modificar y almacenar sin tener en cuenta su complejidad interna. El modelo de animación tiene dos partes bien diferenciadas: la construcción de las animaciones, y su reproducción.

### Construcción de animaciones

Una animación es como una película, por lo que su generación se podría enfocar como tal, generando una secuencia de fotogramas. Estos fotogramas podrían ser el conjunto de las visualizaciones de la expresión inicial, cada expresión intermedia generada y la expresión final.

La automatización total del proceso de construcción de las animaciones es posible, ya que la producción de las visualizaciones de cada paso de ejecución es automática. Pero dicha automatización elimina cualquier decisión de los usuarios sobre su contenido –p.ej., dependiendo de los pasos de ejecución que se muestren, un mismo programa podría utilizarse para visualizar el funcionamiento de la recursividad o el recorrido en profundidad de árboles binarios–. Si la construcción de la animación fuera totalmente automática las animaciones serían siempre las mismas, imposibilitando su adaptación a otros fines. Por esto se optó por un enfoque semiautomático, dando responsabilidad al usuario en los aspectos semánticos de las animaciones y automatizando el resto de procesos.

Según este enfoque, Naharro-Berrocal et al. [148, 149] consideran las animaciones como secuencias de visualizaciones discretas. Como cada paso de ejecución genera una visualización discreta, la generación de la totalidad de pasos de ejecución se realiza de forma automática, dejando al usuario decidir el tipo de paso de ejecución a realizar. La posibilidad de elegir los pasos de ejecución que se mostrarán en la animación se facilita de dos formas: mediante el modelo de ejecución, que permitía cambiar el modo

de avance durante la ejecución del programa, y por lo tanto decidir qué visualizaciones estáticas se generan; o posteriormente, dejando al usuario seleccionar de entre todas las visualizaciones generadas, aquellas que formarán parte de la animación.

Con una interfaz sencilla, el entorno muestra al usuario las visualizaciones generadas y el usuario selecciona aquellas que cree más convenientes (véase la figura 3.7). Dicha interfaz tiene dos posibilidades; en ambos casos la selección de las visualizaciones se realiza marcando un *checkbox* asociado a cada una. La primera posibilidad de selección consiste en mostrar las visualizaciones estáticas a tamaño original en una ventana; como el tamaño y la cantidad de las visualizaciones es variable, esta ventana está dotada de una barra de desplazamiento de forma que se puede acceder a cualquier visualización. Por otro lado, para poder ver una mayor cantidad de visualizaciones, se les ofrece a los usuarios la posibilidad de trabajar con versiones reducidas de las mismas.

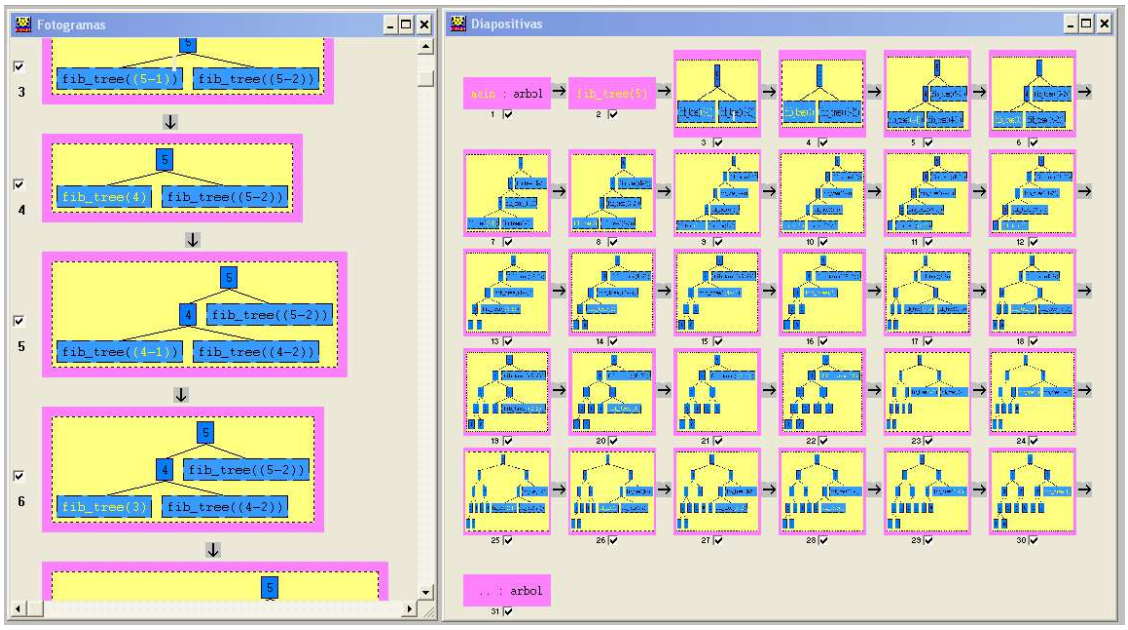



Figura 3.7: Interfaz para la selección de visualizaciones estáticas que formarán parte de la animación. Se muestran las dos posibilidades: en la parte izquierda, en la ventana de "*Fotogramas*" aparecen las visualizaciones a tamaño original, en la parte derecha en la ventana de "*Diapositivas*" aparecen las versiones reducidas

### Reproducción de animaciones

Otro punto distinto es la reproducción de estas animaciones. Siguiendo con la metáfora del documento ofimático, el propio entorno será capaz de reproducir las animaciones, de forma que estas se encuentran totalmente integradas en el entorno de programación. Las animaciones se podrían tratar como películas, pudiéndose pensar en la idea del reproductor de vídeo (véase la figura 3.8), con posibilidad para elegir la velocidad de reproducción, cambiar el sentido, trasladarse al primer o último paso, así como ir un paso adelante o atrás.

Sin embargo, desde el punto de vista educativo, la dependencia total del entorno de programación no es tan ventajosa. Una animación de un programa sirve para más cosas que ver el comportamiento inmediatamente después de hacer un programa. También puede servir para mostrar el comportamiento en clase, o incluso generar animaciones accesibles en otros entornos, como la Web. Así, Naharro-Berrocal

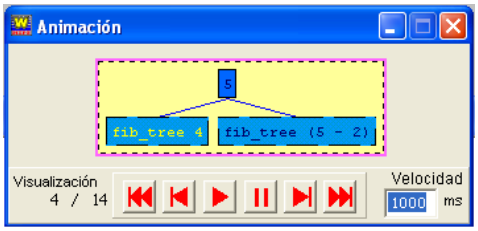



Figura 3.8: Interfaz de reproducción de animaciones integrado en el entorno

et al. [147] describieron la extensión de WinHIPE para generar las animaciones incrustadas en una página Web (véase la figura 3.9), con la ventaja de poder asociar contenidos textuales, aunque estos tuvieran que ser editados directamente por el usuario con alguna aplicación específica de edición de código HTML.

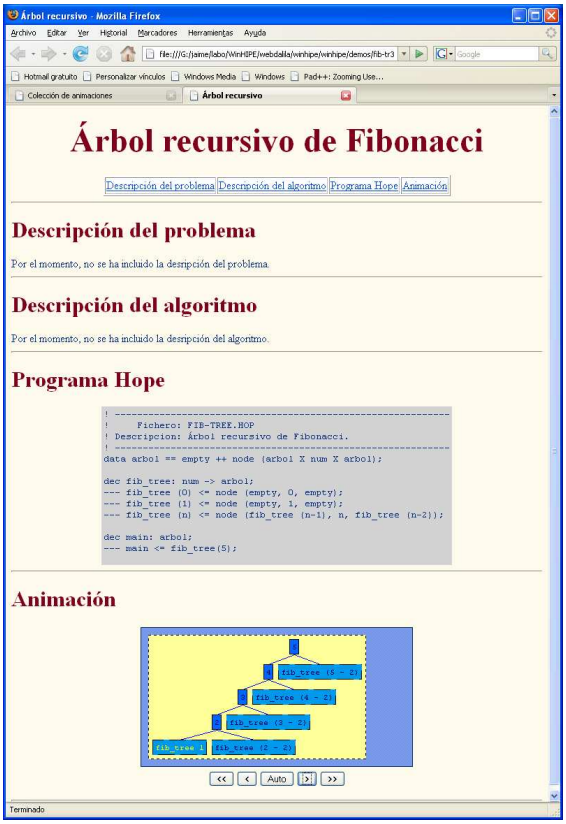



Figura 3.9: Interfaz de reproducción de animaciones en la Web

En resumen, el modelo de animación permite crear animaciones como secuencias de visualizaciones discretas de los distintos estados por los que pasa la ejecución del programa. Esta tarea es simple, ya que consiste únicamente en seleccionar aquellas visualizaciones que se quiere que aparezcan en la animación. Por otro lado, la reproducción de animaciones se puede hacer de forma integrada en el propio entorno o de forma independiente integradas en páginas Web.

## 3.2. Ubicación de las aportaciones de la tesis

En esta sección haremos una breve introducción a las aportaciones realizadas en esta tesis, ubicándolas en el marco de trabajo previamente descrito. Realizamos aportaciones en el *modelo de visualización estática*, y fundamentalmente en el *modelo de animación*.

Uno de los objetivos principales es que la tarea de construcción de animaciones de programas funcionales no requiera apenas esfuerzo por parte del usuario. De esta forma conseguiremos que las animaciones se integren más fácilmente en el ámbito educativo. La figura 3.10 muestra un esquema de la integración de los modelos descritos en secciones anteriores y las aportaciones de esta tesis a dichos modelos.

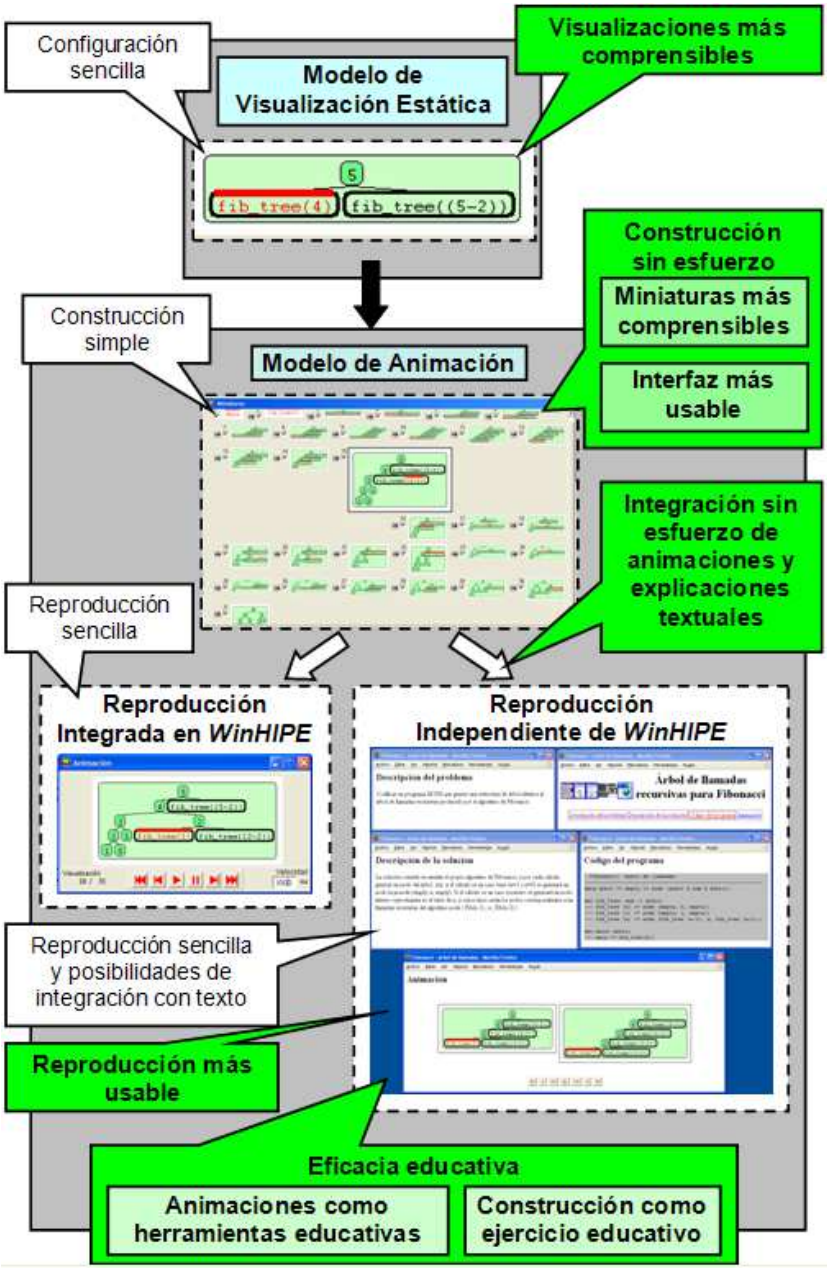



Figura 3.10: Ubicación de las aportaciones de esta tesis (en color verde) en los modelos visualización estática y animación de programas funcionales en WinHIPE.

Hemos atacado el problema tanto desde el punto de vista interactivo como de su uso educativo. La construcción de animaciones de programas funcionales se realizará con la interfaz provista por el entorno de programación WinHIPE. Podemos obtener una panorámica tanto del entorno como de las mejoras realizadas en las publicaciones [169] y [236]. Desde el punto de vista *interactivo*, la interfaz debe ofrecer al usuario información suficiente sobre la ejecución, y herramientas para que pueda construir la animación. Por otro lado, las animaciones generadas por el entorno deben tener un uso educativo. Desde el punto de vista del *uso educativo*, las animaciones de programas funcionales deben ser herramientas que favorezcan el aprendizaje de los estudiantes, pero no sólo el producto final –las animaciones–, sino también el propio proceso de construcción, que puede ser una experiencia educativa provechosa para los estudiantes.

### 3.2.1. Aspectos interactivos en la construcción de las animaciones

Como hemos explicado en el apartado 3.1.3, el marco de trabajo original ofrece un procedimiento de construcción animaciones bastante simple. Tan sólo consiste en seleccionar las visualizaciones estáticas que formarán parte de la animación. Dicha selección se realiza mediante versiones reducidas de las mismas que permitían tener una visión más global de la ejecución del programa. Hicimos una evaluación general del entorno (publicación [135]) y, aunque obtuvimos resultados alentadores con respecto a la opinión y actitud de los estudiantes, pudimos observar que algunos no usaban las versiones reducidas de las visualizaciones[1].

Analizamos el marco de trabajo entero (publicación [146]) desde el punto de vista interactivo. Así, descubrimos que las versiones reducidas de las visualizaciones no eran demasiado comprensibles. Para mejorar la comprensión de las miniaturas cambiamos parte del formato de representación de las visualizaciones estáticas (modelo de visualización estática), pero con el objetivo principal de hacer que sus miniaturas correspondientes se entendieran mejor; además, mejoramos el algoritmo utilizado para generar las miniaturas, de forma que estas fueran de mejor calidad (publicaciones [77, 145]).

La construcción de animaciones no sólo depende de comprender cada uno de los pasos de ejecución, sino de tener también una visión global de la ejecución. Para ello estudiamos la adaptación de una técnica de visualización de información (publicación [77]) llamada *Flip-Zoom* [26, 27], y posteriormente desarrollamos una nueva, llamada *R-Zoom*, que permitiera ofrecer una vista global de la ejecución a la vez que permitir acceder a los detalles de un estado de ejecución determinado (publicaciones [228, 230, 232]).

Finalmente, evaluamos el impacto de estas mejoras en el esfuerzo necesario para la construcción de animaciones (publicación [236]).

Detallamos la descripción, el diseño y la evaluación de las aportaciones relacionadas con el aspecto interactivo de la construcción de animaciones en el capítulo 4.

### 3.2.2. Uso educativo de las animaciones y su proceso de construcción

El simple hecho de que un entorno produzca visualizaciones de los diferentes estados de ejecución de un programa no implica que los estudiantes vayan a aprender mejor con dicho entorno. Así, las animaciones generadas por las primeras versiones de WinHIPE –situación inicial descrita en la sección 3.1– se evaluaron en el ámbito educativo [151], no obteniendo ningún resultado que sustentara la hipótesis de que las animaciones ayudaban al aprendizaje.

[1] Ventana de diapositivas, figura 3.7

Por ello, viendo las animaciones como material educativo, hemos modificado ciertos aspectos de sus contenidos así como formato (publicación [229]). Por un lado, aseguramos la posibilidad de integrar contenidos textuales, y mejoramos el formato Web de las animaciones con un diseño centrado en el estudiante. El resultado han sido animaciones fáciles de usar y pedagógicamente eficaces (publicaciones [227, 231]). Por otro, hemos conseguido que las animaciones Web sean reutilizables, independizándolas de quién las crea. Esto, junto con la facilidad de gestión de las colecciones de animaciones, facilita su uso por parte de los profesores.

En cuanto al proceso de construcción de las animaciones, la inclusión de algunas de las aportaciones mencionadas en el apartado anterior produjo resultados prometedores. Como hemos mencionado en el apartado anterior, evaluamos el uso de las animaciones en un entorno educativo (publicación [236]), con las mejoras en cuanto a la calidad de las miniaturas, así como la adaptación de la técnica Flip-Zoom. Aunque los estudiantes opinaban que el proceso de construcción no requería apenas esfuerzo, no mejoraron en su aprendizaje.

Tomando en consideración las nuevas animaciones, al cambiar sus contenidos, hemos tenido que transformar su proceso de construcción. Basándonos en el proceso anterior, junto con las aportaciones antes mencionadas, aseguramos que tanto la configuración de la apariencia, como la integración de explicaciones textuales en las animaciones se realizarán de forma sencilla. Con ello hemos conseguido que el proceso de construcción sea tan sencillo, que el estudiante se centre más en los conceptos a aprender, que en el propio proceso de construcción de la animación. Hemos comprobado que este proceso, en sí mismo, también aporta mejoras en el aprendizaje de los estudiantes (publicaciones [227, 231]).

Detallamos la descripción, el diseño y la evaluación de las aportaciones relacionadas con el uso educativo de las animaciones y su proceso de construcción en el capítulo 5.

# Capítulo 4

# Visión global y detallada de la ejecución de programas funcionales

En el modelo de animación descrito en el capítulo 3, las animaciones se construyen a partir de secuencias de visualizaciones discretas, correspondiendo cada una de estas a un estado concreto de la ejecución del programa[1]. Pero los programas pueden llegar a generar gran cantidad de visualizaciones. Por lo tanto, el objetivo es permitir al usuario seleccionar aquellas que crea más convenientes para formar parte de la animación, sin que ello suponga una carga de trabajo excesiva.

La animación va a ilustrar la ejecución del programa; por lo tanto, el usuario necesitará una visión global de los estados por los que ha pasado dicha ejecución. Para ello, trataremos de mostrar simultáneamente el máximo número de visualizaciones posibles, adoptando una idea del estilo del clasificador de diapositivas de las típicas aplicaciones ofimáticas de presentaciones (véase figura 4.1). Precisamente, el punto de partida de esta tesis se sitúa en una aproximación reduciendo cada una de las visualizaciones y colocándolas con una distribución similar (véase figura 4.2).

Pero el usuario debe seleccionar las visualizaciones que van a formar parte de la animación, y para ello debe comprender el contenido de cada una de ellas. En la figura 4.2 vemos las visualizaciones reducidas (miniaturas, para el resto de la tesis) generadas durante la ejecución de un programa que calcula el espejo de un árbol binario. En esta figura podemos intuir la estructura de los árboles, aunque hay otros detalles que no se muestran. Sin embargo, si el programa fuera más complicado, o el texto jugara un papel más importante, es evidente que sería muy difícil comprender el contenido de las miniaturas.

Como se puede apreciar en la figura 4.2, imitar a los clasificadores de diapositivas no es una solución muy apropiada, fundamentalmente porque los elementos con los que estos trabajan, diapositivas de presentaciones, no tienen las mismas características que las visualizaciones generadas por *WinHIPE*.

[1] En este capítulo, no tiene sentido diferenciar entre programa o algoritmo, ya que trabajamos con el resultado de sus ejecuciones sin tomar en cuenta el grado de abstracción de sus contenidos.

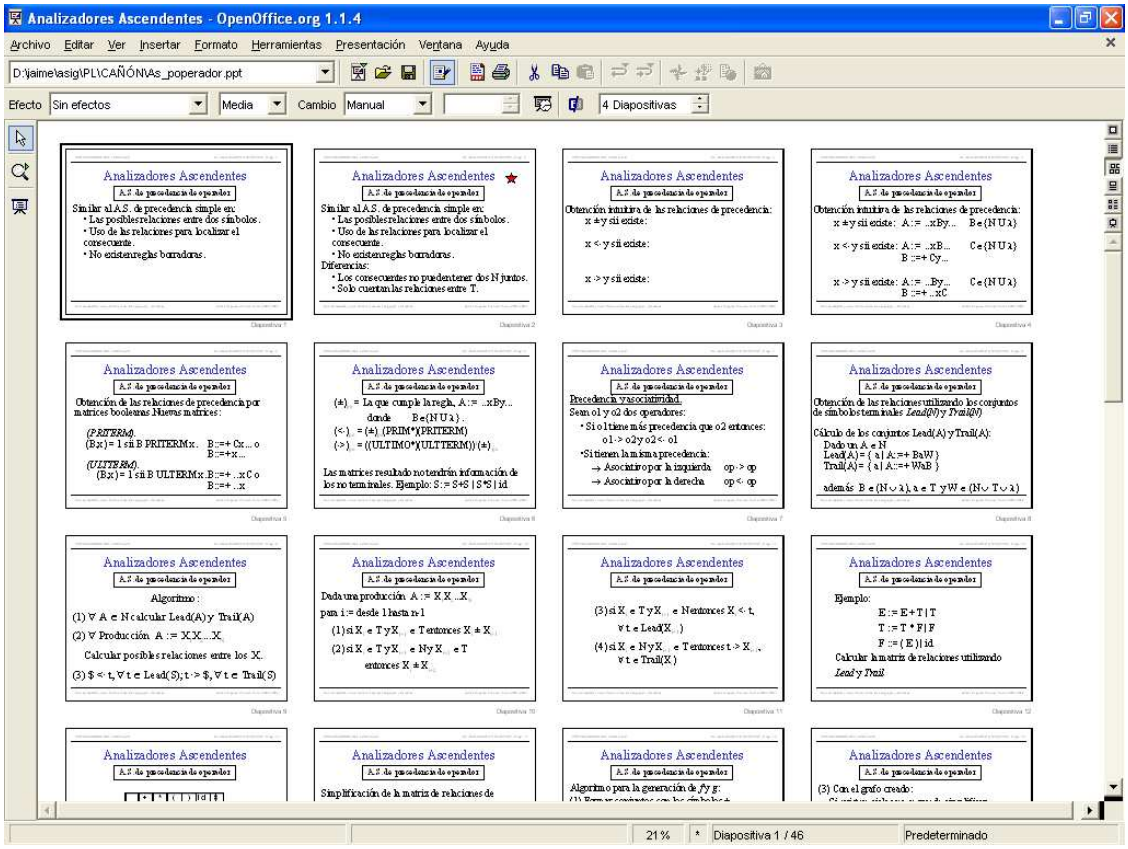



Figura 4.1: Captura del clasificador de diapositivas de OpenOffice Impress

En la tabla 4.1 comparamos las distintas características de las diapositivas y las visualizaciones. Así, en cuanto a *formas y tamaños*, mientras las diapositivas son todas de la misma forma (rectangulares) y tamaño, las visualizaciones son totalmente distintas unas de otras, podrían ser tanto cuadradas como rectangulares horizontales o verticales, y el tamaño dependería de la expresión visualizada. La *estructura del contenido* también cambia; las diapositivas de presentaciones tienen estructuras muy similares, por ejemplo: título, subtítulo y contenido; sin embargo, las visualizaciones tienen estructuras muy variadas, que realmente dependen de la naturaleza del programa visualizado. Referente al *tamaño de letra* utilizado; las diapositivas no suelen tener mucho texto y suelen usarse para presentar información a una audiencia, por lo que es común usar tamaños de letra grandes; mientras que, en las visualizaciones pueden existir muchos elementos y son de un uso más variado[2], por lo que el tamaño de letra usado no tendrá restricciones. Finalmente, el *contenido necesario para comprender* el significado de una diapositiva puede ser bastante poco, frecuentemente con el título y las imágenes basta; sin embargo, aunque unas partes de las visualizaciones sean más importantes que otras, para comprender el significado de estas podría ser necesario ver todo su contenido.

Así pues, nuestro problema radica en reducir las visualizaciones, para mostrar la mayor cantidad de ellas simultáneamente, sin que se vean afectadas significativamente en su comprensión por parte del usuario. La solución que proponemos tiene dos partes. Por un lado nos centraremos en las miniaturas; en la sección 4.1 estudiaremos qué características gráficas de las visualizaciones podemos manipular para conseguir que las miniaturas resultantes sean más comprensibles para el usuario. Por otro lado,

[2]presentación a una audiencia, discusión en grupos reducidos o consulta individual

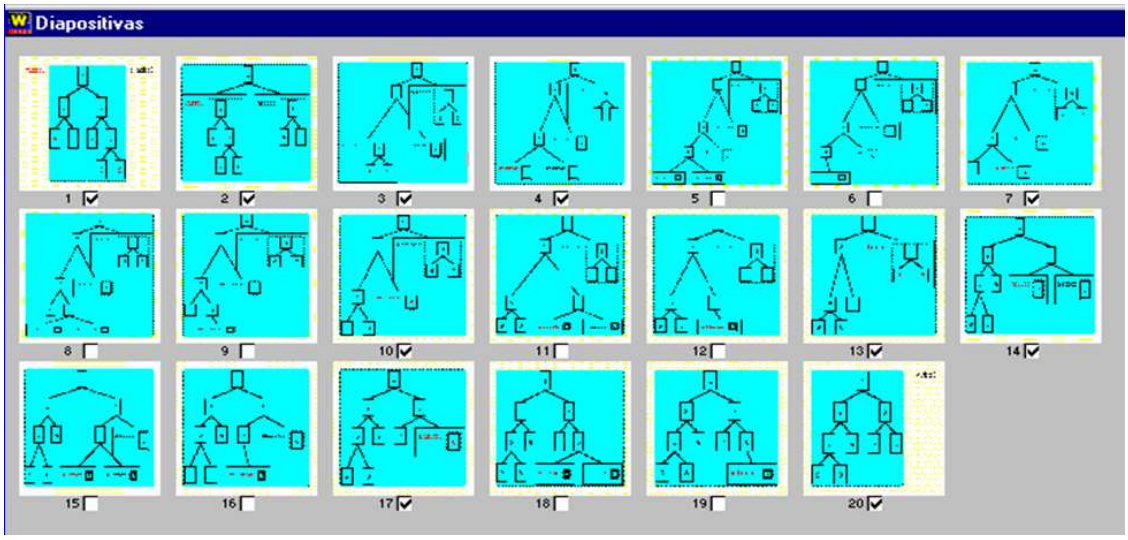



Figura 4.2: Primera versión de la ventana de visualizaciones reducidas (miniaturas) de WinHIPE. La distribución de las visualizaciones reducidas es tabular, teniendo todas las mismas dimensiones y forma

| | ***Diapositivas*** | ***Visualizaciones*** |
|---|---|---|
| ***Formas y tamaños*** | Todas iguales | Distintos |
| ***Estructura del contenido*** | Homogénea: título, subtítulo, y contenido | Heterogénea, depende de la naturaleza del programa visualizado |
| ***Tamaños de letra*** | Grandes | Grandes o pequeños |
| ***Contenido necesario para comprenderlas*** | Poco, título e imágenes | Variado: desde sólo el redex, hasta prácticamente toda la visualización |

Tabla 4.1: Comparativa de características entre diapositivas de presentaciones y visualizaciones de programas

somos conscientes de que el concepto de comprensión depende en gran medida del usuario, por ello, es probable que a pesar de hacer más comprensibles las miniaturas, los usuarios quieran verlas con mayor detalle. En la sección 4.2 describimos *R-Zoom*, una técnica de visualización de información que permite mezclar las miniaturas con visualizaciones mostrando mayor detalle, de esta forma mantenemos la visión global de la ejecución a la vez que ofrecemos una visión más detallada de la miniatura en la que el usuario está interesado en cada momento. Por último, en la sección 4.3 evaluamos el impacto de estas mejoras en el esfuerzo que los usuarios deben dedicar a construir animaciones.

## 4.1. Visualización de miniaturas

Una miniatura es más comprensible cuanto más *se parezca* a la visualización original. Por lo tanto, nuestro objetivo será mantener en las miniaturas tanta información de la visualización original como sea posible. En la figura 4.2 se puede observar la ventana de miniaturas en su estado inicial. La calidad de las miniaturas está fundamentalmente condicionada por dos factores: las dimensiones de las miniaturas y el algoritmo utilizado para la reducción de las visualizaciones.

En primer lugar nos centraremos en las dimensiones de las miniaturas. Como podemos observar

en la figura 4.2, todas las miniaturas son cuadradas y con las mismas dimensiones, a pesar de que las visualizaciones originales no tienen la misma relación de aspecto ni tamaño. Esto se debe a que la distribución de las miniaturas en la ventana es de forma tabular y a que se optó por homogeneizar las dimensiones de cada celda de la tabla. Así, se aprovechaba al máximo el espacio disponible de cada celda de la tabla, evitando el cálculo de los tamaños óptimos de dimensiones de cada fila y columna de la tabla, que es muy complejo y ralentizaría el sistema. En segundo lugar se encuentra el algoritmo utilizado para la reducción de las visualizaciones. Este algoritmo ofrece una solución muy rápida, factor importante en los sistemas interactivos como este, pero de baja calidad.

La solución que proponemos para mejorar la comprensión de las miniaturas, modifica la decisión sobre la relación de aspecto (apartado 4.1.1), añade una característica tipográfica que resalta las partes importantes de las visualizaciones (apartado 4.1.2), y mejora el algoritmo de reducción usado para generar las miniaturas (apartado 4.1.3). A continuación detallamos cada uno de estos tres aspectos.

### 4.1.1. Mantenimiento de la relación de aspecto de las visualizaciones

Los contenidos de las miniaturas de la figura 4.2 son los diferentes estados de ejecución de un algoritmo que calcula el espejo de un árbol. Como es lógico, el árbol se irá modificando, cambiando de un estado a otro su forma, profundidad o anchura. Sin embargo, en la figura 4.2 todas las miniaturas tienen la misma forma (cuadradas) y tamaño, aunque sepamos que no es así en sus visualizaciones originales. En la solución mostrada en la figura 4.2 no se está respetando la *relación de aspecto*.

Llamamos *relación de aspecto* a la proporción existente entre las dimensiones de anchura y altura de una visualización. La relación de aspecto informa sobre el contenido, p.ej., si se está trabajando con árboles, una miniatura vertical indica que el árbol puede tener más profundidad que anchura. Si cambiáramos la relación de aspecto eliminaríamos esta información, o por lo menos obligaríamos al usuario a emplear más tiempo en obtenerla, además de deformar los componentes tanto textuales como gráficos, véase la figura 4.3.

Por lo tanto, siguiendo el principio de hacer las miniaturas lo más parecidas a la visualización original, mantendremos la relación de aspecto de estas en sus miniaturas. Así conseguiremos miniaturas más comprensibles.

### 4.1.2. Resaltado de las partes importantes de una visualización

No todas las partes de una visualización son igual de importantes. De hecho, en nuestro modelo de ejecución de programas funcionales, la parte más importante de una expresión intermedia es el *redex*, que es la subexpresión a evaluar en el siguiente paso de ejecución.

El *redex* señala exactamente el punto actual en que se encuentra la ejecución del programa. Así, destacando el redex, aumentaremos la información que aporta la visualización, y por lo tanto su comprensión. Es verdad que el usuario podría averiguar la misma información sin destacar el redex, pero debería hacer una comparación exhaustiva de una visualización y la siguiente para poder detectar los cambios producidos. Si a esto añadimos la reducción de dichas visualizaciones, la tarea de comparación de dos estados de ejecución sería bastante costosa. Mostramos un ejemplo en la figura 4.4.

Velázquez [235] describe un método para resaltar partes importantes de las visualizaciones mediante el "resumen" parcial de las expresiones que representan, véase un ejemplo en la figura 4.5. Nosotros nos centraremos en un método más visual que se puede utilizar tanto en conjunción como independientemente de estas *expresiones resumidas*.

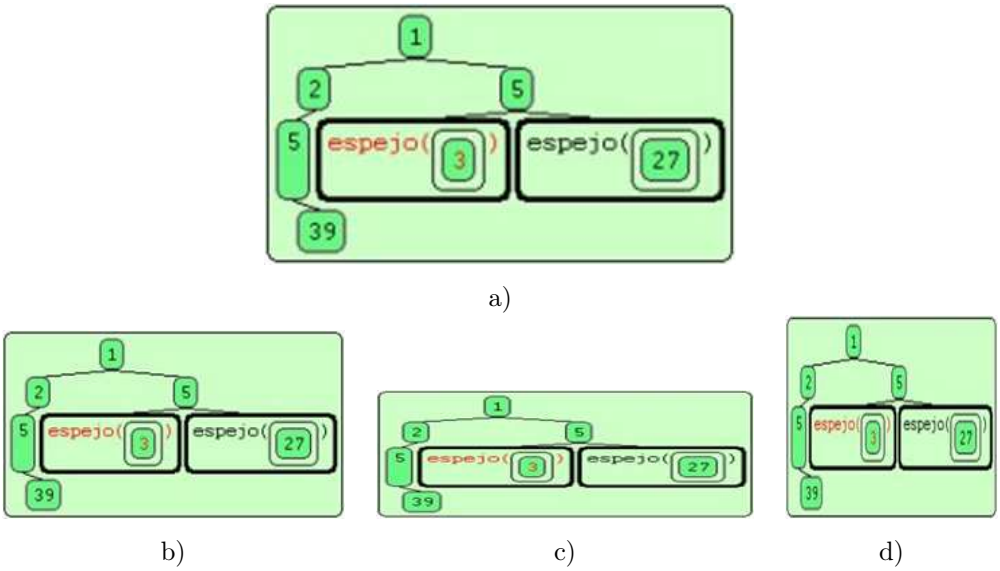



Figura 4.3: Efectos del cambio en la relación de aspecto. Se puede apreciar que la miniatura donde hemos mantenido la relación de aspecto (b) es más parecida a la visualización original (a), y por lo tanto más fácil de interpretar que las otras dos, donde hemos modificado la relación de aspecto en sus componentes vertical (c) y horizontal (d)

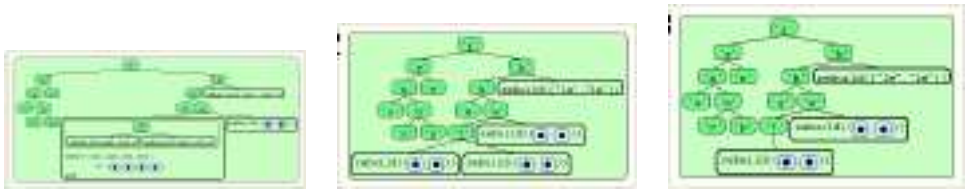
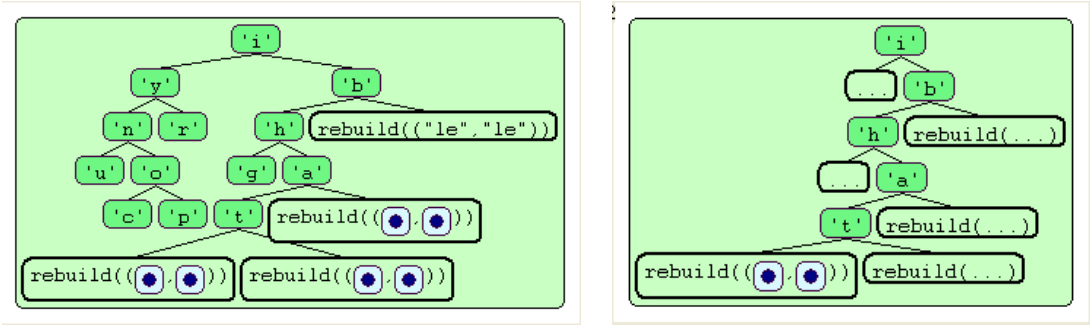


Figura 4.4: Tres miniaturas consecutivas, la tarea de detectar el lugar donde se producen los cambios es bastante costosa

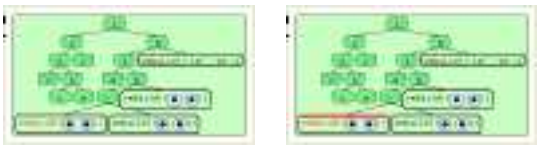
Velázquez y Presa [239] expusieron la idea de destacar con colores diferentes los contenidos textuales del redex. Aunque el efecto es visible en las visualizaciones a tamaño original, véase el texto en color rojo de la figura 4.3a, la comprensión de las miniaturas no ha mejorado mucho, como hemos podido ver en la figura 4.2. La causa principal del problema se encuentra en el proceso de reducción de las visualizaciones. Esta reducción puede difuminar el texto destacado, anulando el efecto deseado (véase figura 4.6a).

De nuevo, seguimos la idea de mantener al máximo la información de las visualizaciones originales en sus respectivas miniaturas. El proceso de reducción atenúa inevitablemente los redex de las miniaturas. Por lo tanto, para conseguir redex más visibles en las miniaturas, habrá que destacarlos más en las visualizaciones originales. Para ello, hemos decidido añadir un elemento gráfico a los redex de las visualizaciones. Dicho elemento es una línea horizontal por encima del redex y en su mismo color. Esta línea apenas afectará a los elementos colindantes y ayudará a detectar el redex tanto en visualizaciones como en miniaturas (véase figura 4.6b).




a) b)

Figura 4.5: Resumen de expresiones funcionales, la figura (a) muestra la expresión completa, la (b) muestra la expresión resumida



a) b)

Figura 4.6: Efecto del redex destacado sin (a) y con (b) la línea superior

### 4.1.3. Algoritmo de reducción de visualizaciones

Finalmente, después de mantener la información de la relación de aspecto y realzar el redex de las visualizaciones, nos queda tratar de mantener al máximo sus contenidos gráficos (las imágenes propiamente dichas). Dichos contenidos son manipulados por el algoritmo utilizado para reducir las imágenes, y según sea la calidad ofrecida por el algoritmo, así serán de comprensibles las miniaturas generadas. En primer lugar revisaremos los distintos algoritmos de reducción de imágenes existentes, y a continuación, expondremos la solución que hemos adoptado.

#### Algoritmos de interpolación para la reducción de imágenes

Dentro de las distintas transformaciones que se pueden aplicar a las imágenes, se encuentran las transformaciones geométricas, como la rotación o el escalado. Los algoritmos de reducción de imágenes forman parte de aquellos dedicados al escalado.

Estos algoritmos se fundamentan en la teoría matemática de la interpolación. Así, los píxeles de las imágenes escaladas se calculan como interpolación de los originales. La idea básica es calcular una aproximación de los píxeles de la imagen original como un polinomio. El rendimiento de estos métodos está condicionado por su capacidad a la hora de generar las aproximaciones [219], así cuanto más complejo sea el cálculo mayor coste computacional. Existen multitud de algoritmos de interpolación en la literatura sobre procesamiento digital de imagen (véanse Baxes [13], Castleman [43], Jähne [100], o Jähne et al. [101]). Nosotros los resumiremos en tres tipos: interpolación de grado 0, interpolación de grado 1 o lineal, e interpolación de grado superior.

Los métodos que ofrecen mejores aproximaciones, y por tanto dan resultados de más calidad son los de *interpolación de grado superior*. Estos tratan de aproximar los píxeles de la imagen original a una representación polinómica de grado mayor que uno (polinomios de Lagrange, Hermite, B-splines, etc).

Una vez que se tiene dicha representación basta con aplicar un factor de escala al polinomio y evaluarlo para obtener los nuevos valores de los píxeles. Debido a la complejidad de los cálculos a realizar, estos métodos son lentos pero, como hemos dicho anteriormente, ofrecen una calidad muy alta [100]. Suelen utilizarse en el tratamiento de imágenes médicas, fotografía digital o imágenes obtenidas de satélites, donde hasta el menor de los detalles es importante.

En el otro extremo se encuentran los métodos de *interpolación de grado 0*, también llamados *interpolación por vecindad*. La idea básica es que los píxeles de la imagen escalada, son los más cercanos a los de la imagen original; en última instancia, esto se implementa replicando o eliminando píxeles de la imagen original. Los resultados de este tipo de métodos tienen una calidad muy baja, apareciendo el conocido efecto *aliasing*, pero son muy rápidos puesto que los cálculos son muy simples [43].

Finalmente se encuentran los métodos de interpolación lineal, que suponen un punto medio entre los dos anteriores [101]. El polinomio obtenido como aproximación de los píxeles de la imagen original es de grado uno. A la hora de reducir imágenes, esto significa que cada píxel de la imagen reducida es la media aritmética de los píxeles más cercanos en la imagen original. El efecto principal de este método es el suavizado de bordes que produce inevitablemente el cálculo de la media. La figura 4.7 muestra un ejemplo de los tres métodos de interpolación mencionados.

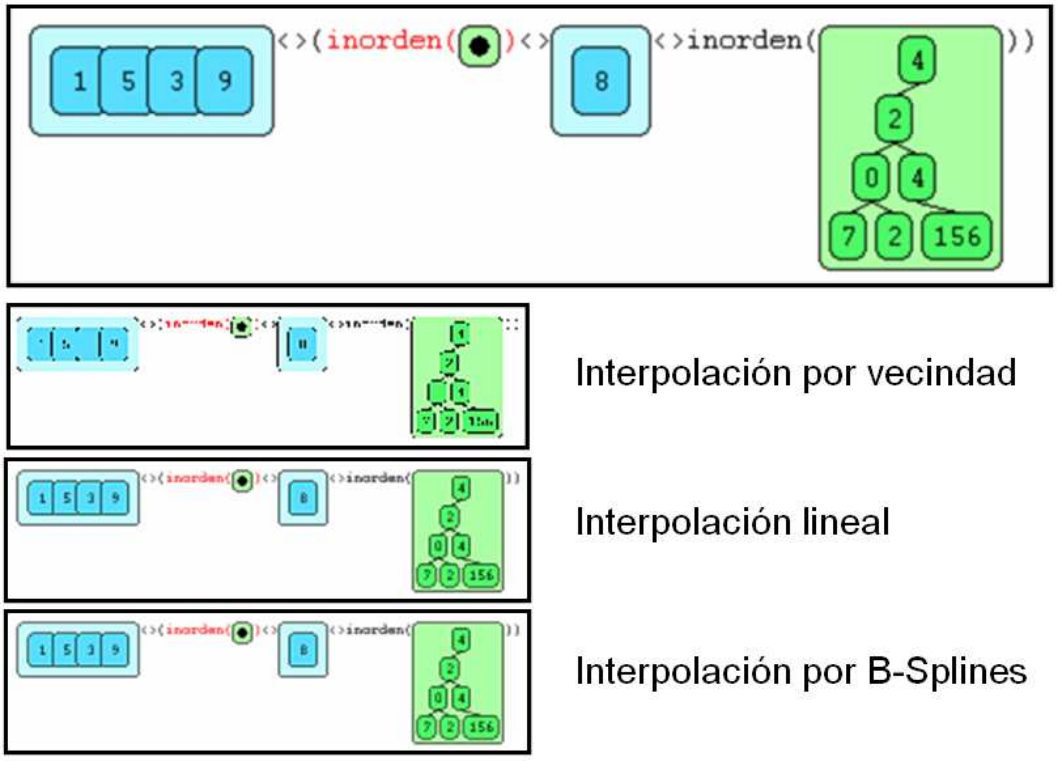



Figura 4.7: Ejemplo de reducción utilizando los tres métodos de interpolación. La imagen superior corresponde a la imagen original, mientras que las inferiores corresponden a una reducción del 50 % del tamaño con cada uno de los métodos mencionados

**Selección del algoritmo de interpolación**

Las dos características principales de los algoritmos anteriormente expuestos –calidad y tiempo de cálculo– son criterios importantes para nosotros. Por un lado es necesario, como mínimo, que el usuario pueda intuir si una miniatura representa a una visualización que debe estar en la animación o no; esto se consigue generando miniaturas de alta calidad. Por otro lado, puesto que este entorno es un sistema interactivo, el tiempo invertido en crear las miniaturas no debe ser excesivo; de lo contrario, el usuario abandonará la tarea o decidirá realizarla de otra forma, haciendo inútil la generación de miniaturas de alta calidad. Por lo tanto, proponemos una solución de compromiso entre la calidad y la velocidad de cálculo. Dicha solución está representada por los métodos de interpolación lineal, que ofrecen una calidad de resultados suficiente para nuestro dominio de trabajo (véase figura 4.7), en un tiempo razonable.

Implementamos un primer prototipo en Java; los dos algoritmos básicos ofrecidos por este lenguaje eran interpolación por vecindad e interpolación lineal, llamados en Java *scale replicate* [137] y *scale area averaging* [136] respectivamente. Como hemos dicho anteriormente, utilizamos el método de interpolación lineal. Tras hacer las primeras pruebas, detectamos cierta lentitud en este algoritmo, así que analizamos en profundidad su funcionamiento para poder detectar el origen de este efecto. A continuación explicamos los pasos de los que se compone este algoritmo:

1. Calcular la proporción entre el tamaño original y el final.

2. Si la proporción no es un número entero, es decir, si el tamaño original no es múltiplo del final hacer:

   *a*) Calcular las dimensiones mínimas que sean múltiplos del tamaño final y mayores que el tamaño original.

   *b*) Crear una imagen intermedia resultado de ampliar la imagen original al tamaño anteriormente calculado, usando el algoritmo *scale replicate*.

3. Como el tamaño de la imagen intermedia (u original) es múltiplo del tamaño de la imagen final, cada píxel de la imagen final se corresponde con una región de NxM píxeles de la imagen intermedia (u original). Cada píxel de la imagen final se calculará como la media de los píxeles intermedios (u originales) de su región correspondiente.

El segundo paso del algoritmo supone una gran pérdida de eficiencia. Supongamos la siguiente situación: tenemos una imagen cuyo tamaño es de `128 x 81` píxeles, y queremos reducirla a `60 x 38` píxeles (un 46.8 % de reducción). El tamaño de la imagen intermedia sería de `180 x 114` píxeles, que es el triple del tamaño final. En la tabla 4.2 mostramos el cálculo del total de píxeles procesados, entendiendo por píxel procesado tanto aquel que se lee de la imagen original o intermedia para el cálculo de la imagen final, como el que se genera en alguno de los pasos 2 ó 3 del algoritmo.

Si consiguiéramos eliminar la utilización de la imagen intermedia, la cantidad de píxeles procesados descendería a **12.648**, lo que representa menos del **23,6 %** de los píxeles procesados con el algoritmo *scale area averaging*. A continuación exponemos dos optimizaciones de este algoritmo. La primera, que llamaremos *Algoritmo A*, se centra en la eliminación de la imagen intermedia; la segunda optimización, que llamaremos *Algoritmo B*, se centra en la optimización del cálculo de las regiones de píxeles originales correspondientes a cada píxel de la imagen final.

| | |
|---|---|
| Uso de imagen original para generar la intermedia | 128x81 = 10.368 px |
| Píxeles generados para la imagen intermedia | 180x114 = 20.520 px |
| Uso de imagen intermedia para generar la final | 180x114 = 20.520 px |
| Píxeles generados para la imagen final | 60x38 = 2.280 px |
| **Total píxeles procesados** | **53.688 px** |

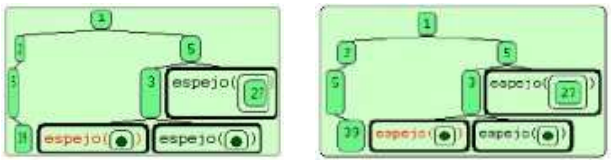
Tabla 4.2: Ejemplo de píxeles procesados con el algoritmo *scale area averaging*

**Algoritmo A: optimización de la interpolación lineal**

La idea principal de esta optimización es evitar la utilización de la imagen intermedia que aparece cuando la dimensión final no es múltiplo de la original. Por lo tanto, el problema se encuentra en cómo identificar la región de píxeles originales correspondiente a cada píxel de la imagen final, cuando esta región no tiene dimensiones medibles en términos de píxeles completos. De aquí en adelante describiremos la solución con una sola dimensión, la extensión a dos dimensiones es trivial.

Continuando con el ejemplo anterior, si queremos reducir una imagen de 128 píxeles de ancho a 60 píxeles de ancho, cada píxel de la imagen final se corresponde con 2,13 píxeles originales ($128/60 = 2,13$). Dicho de otro modo, podemos hacer regiones de 2 píxeles y nos quedaríamos con 8 píxeles sin tratar (resto de la división entera 128/60). Es evidente que no podemos dejar de tratar parte de la imagen original, puesto que la calidad se vería afectada. Luego el problema de calcular las regiones de píxeles originales correspondientes a cada píxel de la imagen final, se transforma en elegir qué regiones van a recibir aquellos píxeles sobrantes.

Existen muchos criterios posibles para asignar los píxeles sobrantes, algunos arbitrarios, como colocarlos en los extremos de la imagen, y otros más lógicos, como repartirlos uniformemente entre las regiones. La figura 4.8 muestra el efecto visual entre ambos criterios. Es evidente que la solución que aportará más calidad global es el reparto homogéneo.



Reparto en los extremos Reparto homogéneo

Figura 4.8: Ejemplo del efecto de los criterios de reparto de píxeles sobrantes entre regiones. Nótese cómo el reparto en los extremos hace que la imagen resultante aparezca deformada en los mismos. En este caso, los nodos de la rama izquierda el árbol son difícilmente reconocibles, y la última parte del nodo derecho de tercer nivel es prácticamente ilegible

Para hacer este tipo de repartos homogéneos, una idea bastante intuitiva es utilizar acumuladores de tipo real, que luego se truncan para poder trabajar en el entorno de los enteros (píxeles completos). El algoritmo implementado con esta idea es realmente una adaptación del *analizador digital diferencial (DDA)* de Sizer [207]. Mostramos el pseudocódigo del algoritmo en la figura 4.9.

Cada iteración del algoritmo trata un píxel de la imagen final, y por lo tanto una región de píxeles originales. Guardamos la cantidad teórica total de píxeles que se deberían haber tratado en `tratadosTeoricamente`, y por cada iteración la incrementamos con el valor teórico del tamaño de

```
variables enteras: TamOriginal, TamFinal, tratadosRealmente, TamRegionActual;
variables reales: TamRegion, tratadosTeoricamente;

tratadosTeoricamente = 0.0;
tratadosRealmente = 0;
TamRegion = TamOriginal / TamFinal;
Mientras (tratadosRealmente < TamOriginal) {
  tratadosTeoricamente = tratadosTeoricamente + TamRegion;
  TamRegionActual = truncar(tratadosTeoricamente - tratadosRealmente);
  ... Acciones a realizar con la región actual ...
  tratadosRealmente = tratadosRealmente + TamRegionActual;
}
```

Figura 4.9: Reparto homogéneo de píxeles con la adaptación del DDA de Sizer [207]

la región, `TamRegion`. El tamaño de la región actual, `TamRegionActual`, se calcula truncando el resultado de restar a `tratadosTeoricamente` los píxeles que realmente se han tratado hasta el momento, `tratadosRealmente`, ya que trabajamos con píxeles enteros. Finalmente, se actualiza el valor de `tratadosRealmente` con el tamaño de la región tratada en esta iteración, `TamRegionActual`. A continuación ofrecemos un ejemplo del funcionamiento de este algoritmo.

Suponemos que la dimensión original es de `17` píxeles, mientras que la final es de `5` píxeles, habiendo así `2` píxeles sobrantes. El tamaño teórico de la región (`TamRegion`) es `3,4`. En la tabla 4.3 mostramos por cada iteración los valores de las variables `tratadosTeoricamente`, `tratadosRealmente` y `TamRegionActual`.

| Número de iteración | | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| `tratadosTeoricamente` | 0,0 | 3,4 | 6,8 | 10,2 | 13.6 | 17 |
| `TamRegionActual` | 0 | 3 | 3 | 4 | 3 | 4 |
| `tratadosRealmente` | 0 | 3 | 6 | 10 | 13 | 17 |

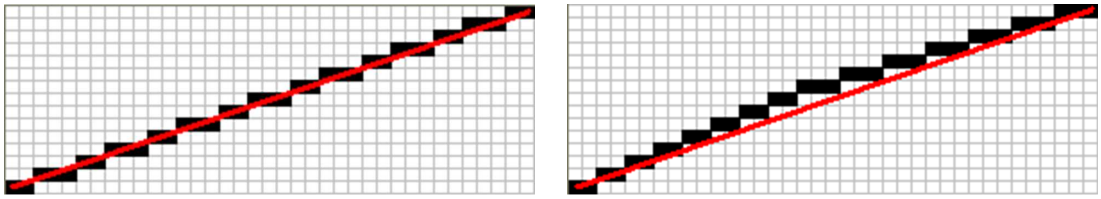
Tabla 4.3: Ejemplo de funcionamiento del algoritmo inicial de reparto homogéneo. Obsérvese cómo la acumulación de trozos de píxeles no tratados llega a ser mayor que la unidad en la iteración número 3, provocando la asignación de uno de los píxeles sobrantes a esa región

La eliminación del uso de la imagen intermedia supone un descenso significativo en el número de accesos a memoria, y por lo tanto, en el tiempo empleado en generar la miniatura. A continuación explicamos una optimización del proceso de cálculo de las regiones correspondientes a los píxeles de la imagen final.

#### Algoritmo B: optimización del algoritmo A para el cálculo de las regiones

El problema del reparto homogéneo de píxeles es análogo al problema de dibujar una recta de pendiente menor que 45° en una cuadrícula. El eje de abscisas representa a los píxeles originales, y el de ordenadas a los píxeles de la imagen final. Los segmentos que conforman la recta son las regiones de píxeles originales que corresponden a cada píxel de la imagen final. La figura 4.10 muestra dos repartos, uno homogéneo que representa una recta y otro no homogéneo que representa, en este caso,

a una línea quebrada.



Reparto homogéneo Reparto no homogéneo

Figura 4.10: Dos ejemplos de repartos, uno homogéneo y otro no homogéneo. En ambos repartos se ha superpuesto la línea recta (en color rojo) que se trata de dibujar en la cuadrícula. Como se puede ver el reparto homogéneo produce una imagen similar a la recta, mientras que el reparto no homogéneo produce una línea quebrada

El algoritmo de Bresenham para el dibujo de rectas [31] es también bastante conocido. Su idea inicial era indicar los movimientos que debía hacer el cabezal de un plóter para dibujar una recta. Este algoritmo tiene una propiedad muy interesante: en el bucle del algoritmo sólo se usan operaciones con números enteros. Por lo tanto, las operaciones a realizar durante el dibujo de la recta son en naturaleza más rápidas que las del *algoritmo A*, que usa operaciones con valores en coma flotante.

De hecho, Field [66] propuso una optimización de la máquina DDA [207] usando el algoritmo de Beresenham como base para eliminar cálculos en coma flotante, cuyo coste computacional es alto. Siguiendo la misma filosofía, exponemos la adaptación del algoritmo de Bresenham para el cálculo de las regiones de la imagen original correspondientes a cada píxel de la imagen final.

El algoritmo de Bresenham es capaz de dibujar líneas de cualquier pendiente y en cualquier dirección. Ambas características dependen de las coordenadas inicial y final de la línea. Sin embargo, nuestro dominio de aplicación es mucho más restringido. Nosotros utilizaremos este algoritmo para la reducción de imágenes. Como dijimos anteriormente, el eje de abscisas (X) representa a los píxeles originales, y el de ordenadas (Y) a los píxeles de la imagen final. Las rectas que tratamos de dibujar siempre parten del origen (0,0), y las coordenadas del punto final cumplen las siguientes reglas:

1. Serán positivas, puesto que no tiene sentido trabajar con tamaños negativos de imágenes.

2. Como reflejan una reducción, los valores originales (X) siempre serán mayores que los finales (Y).

Todas estas restricciones, que definen el conjunto de rectas que podremos dibujar, se representan mediante un sistema de tres inecuaciones cuya solución es el primer octante del plano (véase la figura 4.11). Las rectas correspondientes a este octante se dibujan mediante la parte del algoritmo de Bresenham mostrada en la figura 4.12.

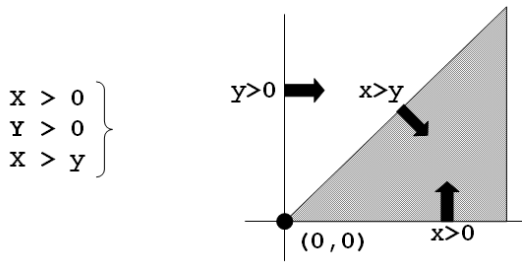



Figura 4.11: El sistema de inecuaciones y su solución nos muestran el tipo de rectas con el que trabajaremos: rectas cuya pendiente es positiva y menor que uno.

```
// Cálculos previos
dx = x1 - x0;
dy = y1 - y0;
ai = (dy-dx)*2;
bi = dy * 3;
d = bi - dx;
// Inicialización, comenzamos en el origen (0,0)
x = y = 0;
do {
  if(d >= 0) then { // movimiento diagonal
    y = y+1;
    d = d + ai;
  } else // movimiento horizontal
    d = d + bi;
  x = x + 1;
  // El punto a dibujar sería el (x,y)
} while(x<dx)
```

Figura 4.12: Parte del algoritmo de Bresenham que dibuja rectas correspondientes al primer octante

La idea de Bresenham consiste en decidir cuándo el siguiente punto a dibujar se encuentra una unidad más arriba en el eje de ordenadas (movimiento diagonal desde el punto actual) o se mantiene en la misma ordenada (movimiento horizontal desde el punto actual), que se corresponden con la parte del `then` y del `else` del algoritmo anteriormente descrito.

Adaptado a nuestro problema, iremos tratando cada píxel original, y el algoritmo de Bresenham nos dirá si pertenece a la misma región, en el caso del movimiento horizontal, o comienza una región nueva, en el caso del movimiento diagonal. Finalmente, nuestro problema es reducir imágenes, por lo tanto trabajamos con dos dimensiones, mientras que el algoritmo de Bresenham sólo nos da información sobre una. La extensión a dos dimensiones es trivial, bastará con anidar un bucle dentro de otro para tratar filas y columnas (véase la figura 4.13).

**Optimizaciones al algoritmo de Bresenham.** Se han dedicado bastantes esfuerzos a optimizar el propio algoritmo de Bresenham. Como este algoritmo tiene aplicación directa en tecnologías gráficas, algunas optimizaciones se realizan desde el punto de vista del hardware. Angel y Morrison [5] desarro-

```
/************** Tratamiento de filas **************/
filas_dx = filas_x1 - filas_x0;    /* Cálculos previos e inicializaciones */
filas_dy = filas_y1 - filas_y0;    /* de variables para filas             */
filas_ai = (filas_dy-filas_dx)*2;
filas_bi = filas_dy * 3;
filas_d = filas_bi - filas_dx;
filas_x = filas_y = 0;
do {
    if(filas_d >= 0) then { /************** Tratamiento de columnas ************/
        columnas_dx = columnas_x1 - columnas_x0;    /* Cálculos previos e      */
        columnas_dy = columnas_y1 - columnas_y0;    /* inicializaciones de     */
        columnas_ai = (columnas_dy-columnas_dx)*2;  /* variables para columnas */
        columnas_bi = columnas_dy * 3;
        columnas_d = columnas_bi - columnas_dx;
        columnas_x = columnas_y = 0;
        do {
            if(columnas_d >= 0) then {
                /************ Region identificada - hacer interpolación *********/
                columnas_y = columnas_y+1;
                columnas_d = columnas_d + columnas_ai;
            } else
                columnas_d = columnas_d + columnas_bi;
            columnas_x = columnas_x + 1;
        } while(columnas_x<columnas_dx)  /** Fin del tratamiento de columnas ***/
        filas_y = filas_y+1;
        filas_d = filas_d + filas_ai;
    } else
        filas_d = filas_d + filas_bi;
    filas_x = filas_x + 1;
} while(filas_x<filas_dx)
/**************** Fin del tratamiento de filas ************************/
```

Figura 4.13: Aplicación del algoritmo de Bresenham al problema de la reducción de imágenes.

llaron una optimización para plataformas de computación paralela. El propio Bresenham [32], Khalida y Kaykobad [202] o Rokne et al [186], desarrollaron optimizaciones para la implementación hardware del algoritmo. Desde el punto de vista de las optimizaciones software, todas se centran en reducir el número de iteraciones necesarias para dibujar la recta. Así, Gardner [72] aprovecha la propiedad de la simetría que tienen las rectas dibujadas por el algoritmo de Bresenham. Otros autores investigaron la posibilidad de calcular más de un píxel por iteración, como Rokne y Wu [187], o Gill [75]. Finalmente, Chen [45] desarrolló una optimización que dibuja rectas a base de replicar sus distintos segmentos.

La adaptación de estas optimizaciones a la reducción de imágenes es compleja. Por lo tanto, primero estudiaremos el rendimiento ofrecido por el algoritmo de Bresenham –nuestro *Algoritmo B*–, después valoraremos la adaptación de las optimizaciones antes mencionadas.

**Estudios comparativos de los tres algoritmos de interpolación: scale area averaging, algoritmo A y algoritmo B**

Como dijimos al principio de la sección, el objetivo es generar miniaturas de calidad suficiente en un tiempo razonable. La calidad viene por el método de interpolación lineal, la velocidad por su implementación. En este apartado vamos comparar las tres implementaciones que hemos expuesto anteriormente, el algoritmo *scale area averaging* y las dos optimizaciones. Ambas optimizaciones estaban pensadas para ahorrar tiempo, tanto por el número de accesos a memoria (Algoritmo A), como por el tipo de operaciones numéricas realizadas durante el cálculo de las regiones (Algoritmo B).

Realizamos dos estudios comparativos, ambos en la misma máquina, una estación Microsoft Windows 2000 Professional, funcionando sobre un Pentium III 550 Mhz con 196 Mb de memoria RAM y 10 Gb de disco duro. La aplicación que ejecuta los algoritmos está programada en Java, el algoritmo *scale-area-averaging* viene implementado en el paquete `java.awt.image`[3], mientras que los otros dos son de implementación propia. La versión de la máquina virtual de Java utilizada en el estudio es la 1.4.2.

**Primer estudio.** En el primer estudio analizamos el tiempo invertido en cargar una colección de imágenes y aplicar a todas ellas el factor máximo de reducción. Este estudio representa la situación inicial de trabajo, donde primero se generan las visualizaciones y a continuación se reducen para mostrarlas al usuario.

**Datos de prueba.** En este estudio hemos utilizado cinco colecciones distintas de visualizaciones. Cada colección posee diferentes propiedades, como el número de visualizaciones, su tamaño (grandes, medianas o pequeñas), su forma (cuadradas o rectangulares) o su contenido (textual o gráfico). Así, las colecciones 1 y 2 tienen visualizaciones gráficas de tamaño grande con formas tanto cuadradas como rectangulares. La colección 3 contiene sólo visualizaciones gráficas grandes y rectangulares. La colección 4 contiene visualizaciones textuales pequeñas y rectangulares. Y finalmente, la colección 5 contiene visualizaciones gráficas de tamaño medio y forma cuadrada.

**Protocolo y resultados.** Debido a que el procesamiento de cada colección no es inmediato, y que no podemos controlar parámetros como la carga del procesador o el acceso a memoria, decidimos repetir los cálculos 50 veces. Así, los resultados con los que trabajamos se calculan como la media aritmética de las 50 medidas por cada colección. La tabla 4.4 muestra estos resultados. En vez de especificar los resultados de los tres algoritmos, mostramos la mejora relativa de ambas optimizaciones con respecto al algoritmo *scale area averaging*.

Como se puede apreciar en la tabla 4.4, ambos algoritmos mejoran substancialmente el comportamiento del algoritmo *scale area averaging*, siendo ambas optimizaciones prácticamente iguales (la diferencia es algo más de 2 %).

**Segundo estudio.** En este estudio analizamos el tiempo invertido en calcular la versión reducida de una única imagen, situación que ocurrirá durante la manipulación de las miniaturas, cuando el usuario decida ampliarlas para verlas más en detalle, o reducirlas para poder ver mayor cantidad de ellas.

[3] http://java.sun.com/j2se/1.5.0/docs/api/java/awt/image/package-summary.html

| | Número de visualizaciones | Tamaño en Kbytes | Algoritmo A | Algoritmo B |
|---|---|---|---|---|
| Colección 1 | 24 | 77,5 | 32,16 % | 36,21 % |
| Colección 2 | 31 | 109 | 23,75 % | 23,77 % |
| Colección 3 | 49 | 191 | 31,58 % | 31,66 % |
| Colección 4 | 18 | 22,3 | 41,56 % | 47,18 % |
| Colección 5 | 15 | 47,4 | 40,72 % | 41,63 % |
| **Media total** | | | **33,95 %** | **36,09 %** |

Tabla 4.4: Tabla de resultados del primer estudio comparativo entre los tres algoritmos. La primera columna identifica la colección de visualizaciones, la segunda el número de visualizaciones en la colección, la tercera el tamaño en Kbytes, y la cuarta y quinta, la mejora relativa al tiempo empleado por el algoritmo *scale area averaging* de los algoritmos A y B respectivamente.

**Datos de prueba.** Hemos trabajado con un rango de porcentajes de reducción que va desde un 95 % hasta un 5 %. En este estudio trabajamos con seis imágenes distintas, tres de ellas genéricas y otras tres generadas con *WinHIPE*.

**Protocolo y resultados.** A pesar de que el tiempo de procesamiento por cada imagen es mucho más rápido que antes, también decidimos repetir los cálculos y trabajar con los promedios de las medidas de las 6 imágenes; así nos aseguramos de que las medidas no se ven afectadas por situaciones anómalas en cuanto al funcionamiento de la plataforma que hemos utilizado. Mostramos los resultados en la figura 4.14.

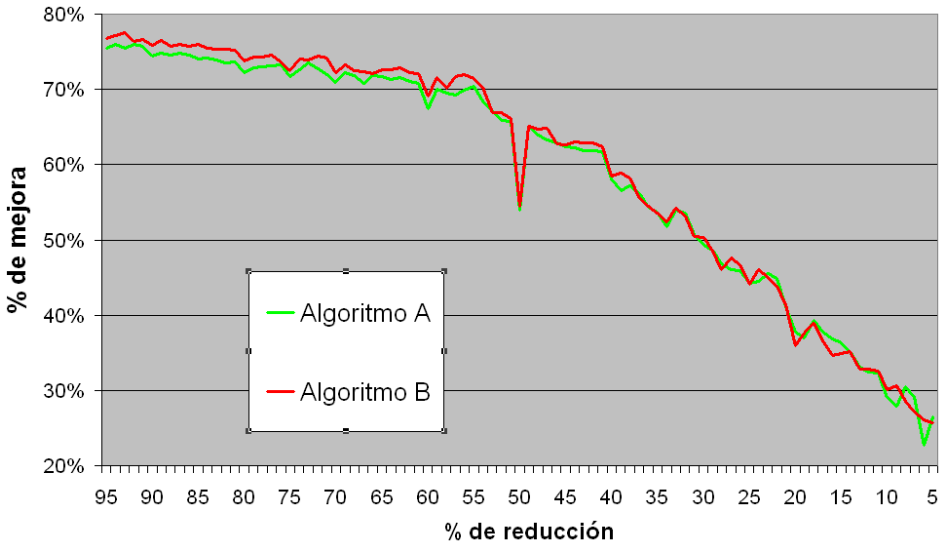



Figura 4.14: Gráfica comparativa de las dos optimizaciones del algoritmo *scale area averaging*. En rojo se muestran los resultados del *Algoritmo A*, en verde los resultados del *Algoritmo B*. El eje X representa el porcentaje de reducción aplicado (el tamaño de la imagen final es un X % del tamaño original). El eje Y representa la mejora relativa con respecto al algoritmo *scale area averaging*.

A la luz de estos resultados caben varios comentarios. En primer lugar, sin distinguir las dos op-

timizaciones, se ve una clara tendencia relacionada con el factor de reducción. La razón fundamental es la eliminación de la imagen intermedia. Para reducciones pequeñas, el tamaño de la imagen intermedia generada prácticamente cuadruplicará[4] al de la imagen original, necesitando numerosos accesos a memoria con el coste en tiempo que esto conlleva. Cuanto más se reduzca la imagen, menor será el tamaño de la imagen intermedia, y por lo tanto, menor el tiempo invertido en los accesos a memoria. Esto explica el excepcional porcentaje de mejora, entre un 70 % y un 77 %, de prácticamente la primera mitad de la escala. A partir del factor de reducción de 45 % (casi la mitad de la imagen original) los tamaños de la imagen intermedia se acercarán al de la original y los nuevos píxeles generados para la imagen intermedia serán pocos, pero habrá que seguir generándola. Así, cuanto más se reduce la imagen final, menor es la mejora de las optimizaciones.

En segundo lugar, la *diferencia entre las dos optimizaciones también depende del factor de reducción*. La razón estriba en el número de iteraciones que debe hacer cada optimización. En el caso del *algoritmo A*, el número de iteraciones depende del tamaño final: cuanto mayor sea este, más restas en coma flotante y truncamientos deberá realizar. En el caso del *algoritmo B*, el número de iteraciones no cambia, sólo cambia el número de veces que bifurca por una rama u otra del `if` que decide cuándo ha terminado una región. El resto de operaciones son similares. Por lo tanto, las diferencias entre ambos algoritmos son ligeramente visibles con factores de reducción pequeños, pero en factores de reducción grandes las diferencias son prácticamente inexistentes. En términos estadísticos, las diferencias existentes entre ambos algoritmos son significativas $Z = -5,241, p = 0,000$, aunque su magnitud no sea relevante, 0,72 % de mejora media del *algoritmo B* respecto al *A*.

**Conclusiones de ambos estudios.** Los resultados muestran claras mejoras de rendimiento de ambas optimizaciones con respecto al algoritmo inicial, *scale area averaging*. Por otro lado, vemos que el *algoritmo B*, es ligeramente mejor que el *algoritmo A*. La razón fundamental es que las operaciones de mayor coste, en cuanto a reducción de imágenes se refiere, son los accesos a memoria. Por lo tanto, ambas optimizaciones son significativamente mejores con respecto a *scale area averaging*. Una vez realizada esta optimización, el número de accesos a memoria es igual en los dos algoritmos; sólo cambia el cálculo de las regiones. Como las operaciones involucradas en este cálculo son claramente más rápidas que los accesos a memoria, la mejora de rendimiento total que aportan es mucho menor, haciendo que el *algoritmo B* sea ligeramente mejor que el *algoritmo A*. Esto lo podríamos extrapolar a las optimizaciones existentes del propio algoritmo de Bresenham. Si las diferencias entre cálculos en coma flotante y cálculos enteros son pequeñas, las diferencias entre sucesivas optimizaciones de Breseham serían aún más pequeñas.

Las mínimas diferencias existentes entre el *algoritmo A* y el resto de optimizaciones que hemos considerado nos conduce a la siguiente conclusión. El esfuerzo de implementación[5] que tendríamos que hacer para integrar el *algoritmo B*, así como sus posibles optimizaciones, no se vería reflejado en la velocidad del entorno al generar las miniaturas. Por lo tanto, optamos por usar el *Algoritmo A*, más sencillo de implementar y con una mejora entre el 34 % y el 40 % con respecto a *scale area averaging*.

[4] El tamaño de la imagen intermedia prácticamente duplicará al original en ambas dimensiones, por lo que en términos de píxeles totales, prácticamente cuadruplicará al original.

[5] WinHIPE está desarrollado en Delphi 5.0

### 4.1.4. Resultado final de las mejoras realizadas en las miniaturas

Las figuras 4.15 y 4.16 muestran el efecto de las mejoras realizadas hasta el momento. Ahora respetamos la relación de aspecto de las visualizaciones en las miniaturas, lo que las hace más comprensibles. Al resaltar el redex en las visualizaciones, permitimos que en las miniaturas se perciba perfectamente. Finalmente, la calidad ofrecida por el algoritmo de reducción de imágenes usado es claramente superior. Estas tres mejoras aplicadas dan como resultado una visión global de las miniaturas mucho más comprensible. Un ejemplo claro es ver cómo en la situación actual podemos percibir claramente, y sin esfuerzo, el flujo de la ejecución del programa.

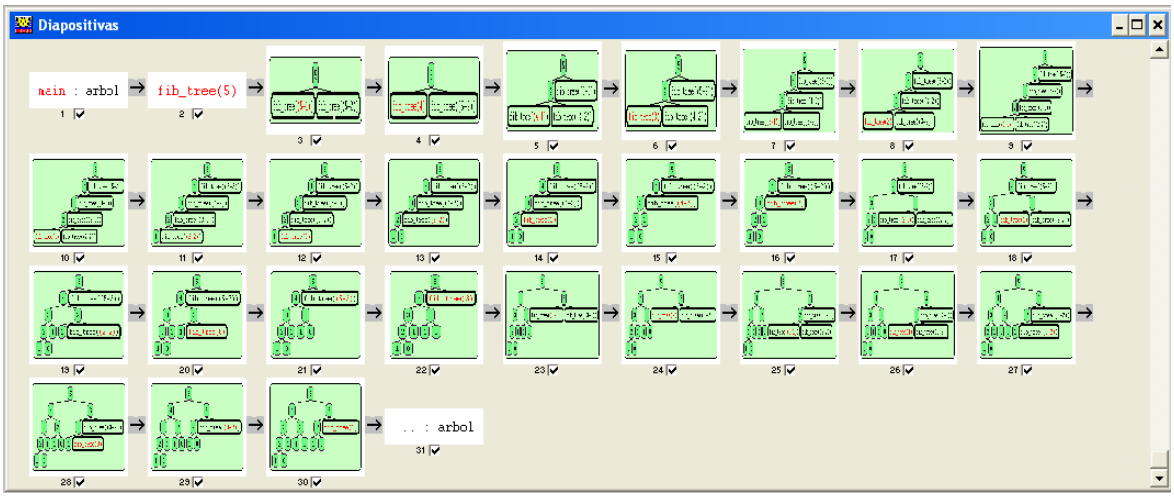



Figura 4.15: Situación inicial de las visualizaciones de miniaturas

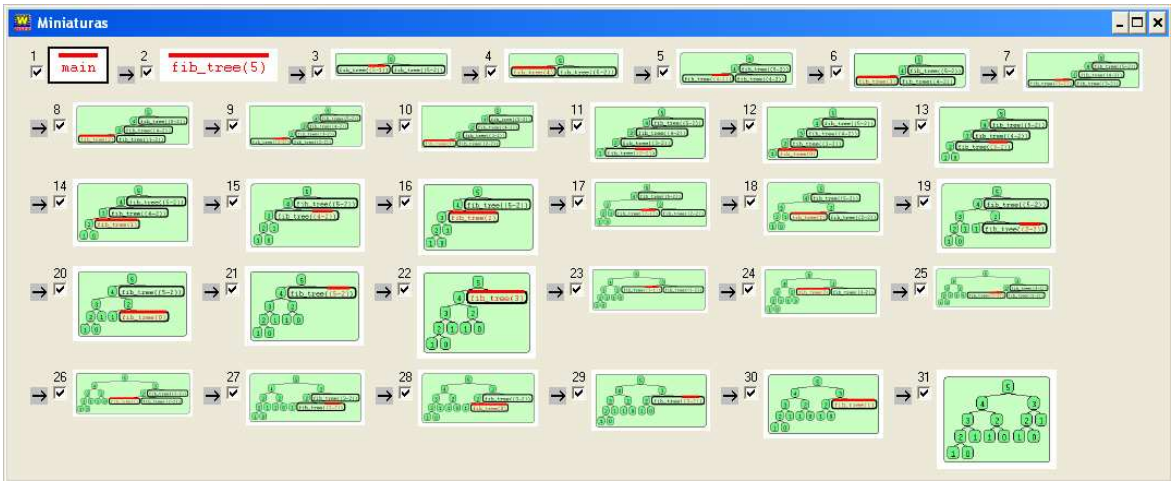



Figura 4.16: Situación actual de las visualizaciones de miniaturas

## 4.2. Selección de miniaturas

Nuestro modelo de construcción de animaciones se basa en la selección de las visualizaciones discretas que formarán parte de la animación. Para ello se trata de ofrecer una visión global de todas las visualizaciones, lo que redundará en una visión global de la ejecución del programa o algoritmo que se quiere animar. Dicha visión global se ha conseguido a través de reducciones de las visualizaciones (miniaturas). En la sección anterior hemos descrito un algoritmo y un conjunto de decisiones que consiguen reducir visualizaciones donde la calidad resultante y el tiempo de cálculo utilizado no entorpecen la tarea del usuario. Sin embargo, como podemos observar en la figura 4.16, hay detalles que no se pueden ver con claridad, y el usuario podría necesitarlos para asegurarse de que la visualización correspondiente debe formar parte de la animación. En esta sección presentamos *R-Zoom*, una técnica de visualización de información que permite mostrar la visualización seleccionada por el usuario en detalle y el resto de visualizaciones en su versión de miniatura, consiguiendo una visión global de la ejecución del programa o algoritmo, y detallada de un paso en particular de dicha ejecución.

A continuación describiremos brevemente los desarrollos previos sobre este problema así como las diferentes técnicas existentes que permiten tener vistas globales y detalladas de información. Después describiremos *R-Zoom*, la técnica que hemos desarrollado para resolver el problema. Y finalmente expondremos los resultados de su evaluación.

### 4.2.1. Desarrollos previos y técnicas de visualización existentes

#### Desarrollos previos

Como ya describimos en el capítulo 3, la primera solución desarrollada fue una ventana que mostraba todas las visualizaciones a tamaño original haciendo uso de la barra de desplazamiento. A pesar de poder acceder a todas las visualizaciones, con esta solución, la visión global es prácticamente nula. Por ello se implementó una solución que mantuviera dos vistas distintas, una con las miniaturas que ofrece una visión global de las visualizaciones, y otra vista que muestra en cada momento una visualización a tamaño original. A pesar del avance significativo que supuso el segundo desarrollo, durante una evaluación general de la herramienta [135], pudimos observar que algunos de los alumnos no trabajaban con dicha solución sino que optaban por utilizar la ventana con la barra de desplazamiento. Así que analizamos en profundidad las distintas alternativas existentes para tratar de dar una solución válida que pudieran utilizar los usuarios; a continuación describimos dicho estudio.

#### Técnicas de visualización global y detallada de información

Existen multitud de técnicas de *visualización de la información*, y su clasificación ha sido una tarea enfocada desde muchos puntos de vista. La mayoría de las taxonomías existentes consideran los datos, de una u otra forma, como una de las características esenciales. Así, Ware [241] clasifica las distintas técnicas según la naturaleza de los datos (entidades, relaciones, atributos, etc); Tory y Moller [222], usan características de alto nivel, como la continuidad del modelo de datos[6], para finalmente clasificar a bajo nivel según características detalladas de los datos como su dimensionalidad, estructura interna, etc.

Otras taxonomías añaden la característica del procesamiento o la manipulación que sufren los datos para ser visualizados. Chi [47] usa su *Data State Model* para clasificar las técnicas según las distintas

[6]Según Tory y Moller, un modelo de datos es continuo si la técnica en cuestión asume que los datos son interpolables

formas que toman los datos durante su procesamiento; Leung y Apperley [127] se centran en las técnicas de distorsión utilizadas para mostrar los datos; mientras que Card y Mackinlay [41] clasifican las técnicas de visualización según la forma de interactuar con los datos visualizados, así como el modo en que se representan.

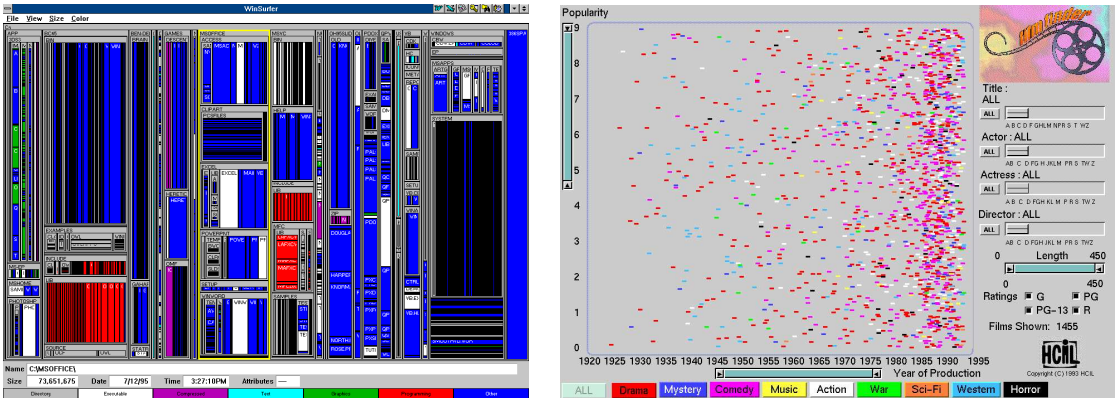
Finalmente, Wehrend y Lewis [243] y Shneiderman [203], utilizan tanto los datos como las tareas que se deben realizar con ellos. Así, se describen tareas como obtener una visión global, hacer zoom sobre ciertos elementos, filtrar, obtener detalles de los elementos, ver relaciones existentes, mantener un histórico de acciones, o extraer conjuntos de elementos que cumplan ciertas condiciones.

Nosotros utilizaremos este punto de vista para el análisis de las técnicas existentes, distinguiendo dos características fundamentales:

**Dominio de aplicación** que son las tareas que ayudan a realizar. En nuestro caso se trata de permitir obtener detalles de las miniaturas mediante zoom, dejando disponible cierta visión global.

**Propiedades de la información que manejan** que puede variar desde el número de dimensiones que tiene, hasta su estructuración interna, p.ej., lineal, en árbol o en grafo. En nuestro caso tratamos con miniaturas, elementos con una representación gráfica propia, de naturaleza unidimensional temporal.

Durante el análisis hemos encontrado técnicas especializadas en jerarquías usando el 100 % del espacio disponible en pantalla, como Tree-Maps [103] (véase la figura 4.17a), o las técnicas llamadas *Dynamic Queries* [206], dedicadas a la exploración de bases de datos mediante controles (véase la figura 4.17b).



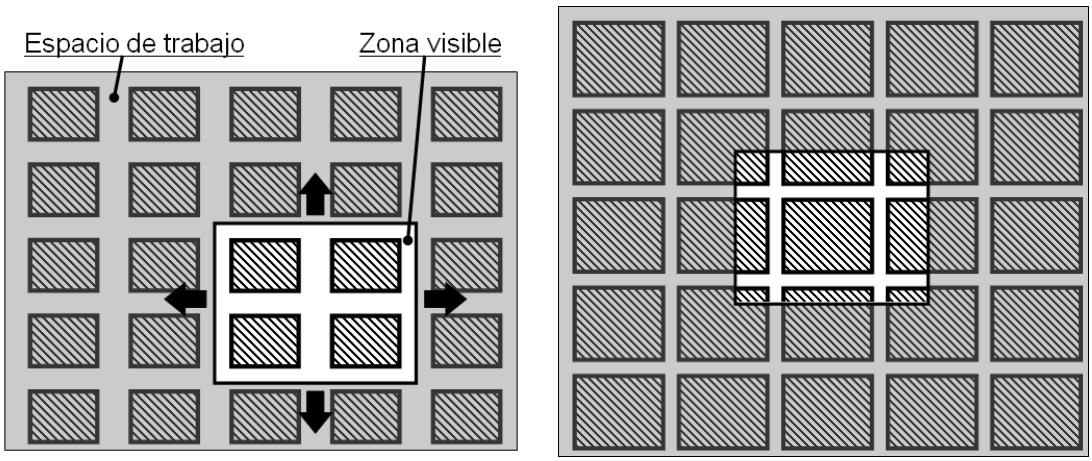

a) WinSurfer [175] ©2004 ACM b) Filmfinder [175] ©2004 ACM

Figura 4.17: Dos ejemplos de técnicas de visualización: a) WinSurfer, una implementación de la técnica *Tree-Map* para representar jerarquías de directorios; b) Filmfinder, implementación de la técnica *Dynamic Queries* para consultar una base de datos de películas.

Ambas técnicas muestran visiones globales y detalladas de los elementos que manejan. Así, *Tree-Maps* muestra una visión global de una jerarquía, mientras que ofrece detalles de los elementos de dicha jerarquía, como el tamaño ocupado por un directorio. *Dynamic Queries* muestra mayor o menor detalle según se ajusten los valores de los controles de visualización. Sin embargo, estas dos técnicas están demasiado centradas en alguna de las dos características que hemos mencionado anteriormente: *Tree-Maps* es específica para estructuras jerárquicas (propiedad de la información que se maneja), y *Dynamic Queries* se diseñó para la exploración de bases de datos (dominio de aplicación).

Nuestro estudio de posibles alternativas se ha centrado en tres familias de técnicas cuyo dominio de aplicación y propiedades de información son más generales, y por lo tanto con más probabilidades de ser aplicadas a nuestro problema. Dichas familias son *Zoom+Pan*, *Overview+Detail*, y *Focus+Context*. El lector interesado puede obtener más información de todas las técnicas en la compilación realizada por Card et al. [42], así como en los libros de Chen [44], Spence [209] o Ware [241].

**Técnicas Zoom+Pan.** Este tipo de técnicas organizan la información distribuyéndola en el espacio de trabajo, permitiendo al usuario manipularla interactuando directamente con ella. Como el espacio de trabajo es más grande que la pantalla, el usuario sólo puede acceder a una parte de él (zona visible) en cada momento. Sin embargo, podrá desplazarse por el espacio de trabajo (*panning*), cambiando la zona visible (véase figura 4.18a), así como acercarse o alejarse (*zooming*) (véase figura 4.18b) de los elementos. El efecto visual de acercamiento y alejamiento se consigue utilizando transformaciones geométricas de escala sobre los objetos. Existen otras variantes de esta técnica, algunos ejemplos son: el *zoom semántico* [172], que cambia el grado de detalle con el que se muestran los elementos en vez de la escala; el *zoom no lineal* que permite acciones como aplicar la escala apropiada para ver determinado objeto [244]; o relacionar cambios de zoom con acciones de desplazamiento por el espacio de trabajo [95].




a) Panning b) Zooming

Figura 4.18: Esquema de interfaces panning y zooming

Perlin y Fox desarrollaron un sistema representativo de esta familia llamado *Pad* [172]. Pad pretendía ser una alternativa a los sistemas de *icono-ventana* usados en otras interfaces. Pad permitía al usuario seleccionar objetos de la interfaz para mostrarse con un grado de detalle mayor (véase la figura 4.19), ya sea mediante ampliación gráfica o mediante zoom semántico. Como añadido, Pad permitía cambiar la representación de datos , p.ej., los datos tabulares se pueden mostrar en forma de tabla, o como un gráfico de barras.

Posteriormente Bederson y Hollan [18] desarrollaron *Pad++*, una extensión del sistema *Pad* hacia un API para el desarrollo de interfaces con *Zoom*, basado en C++ y Tcl/Tk; a día de hoy y tras pasar una recodificación a Java y una reestructuración, se llama Piccolo [17]. Uno de los usos más recientes de técnicas *Zoom+Pan* se puede ver en el nuevo modelo de escritorio adoptado en la última de versión del sistema operativo de Apple *MAC OS X Leopard Sneak Peak*[7].

[7]http://www.apple.com/macosx/leopard/spaces.html (2007) Copyright ©2007 Apple Inc. All rights reserved.

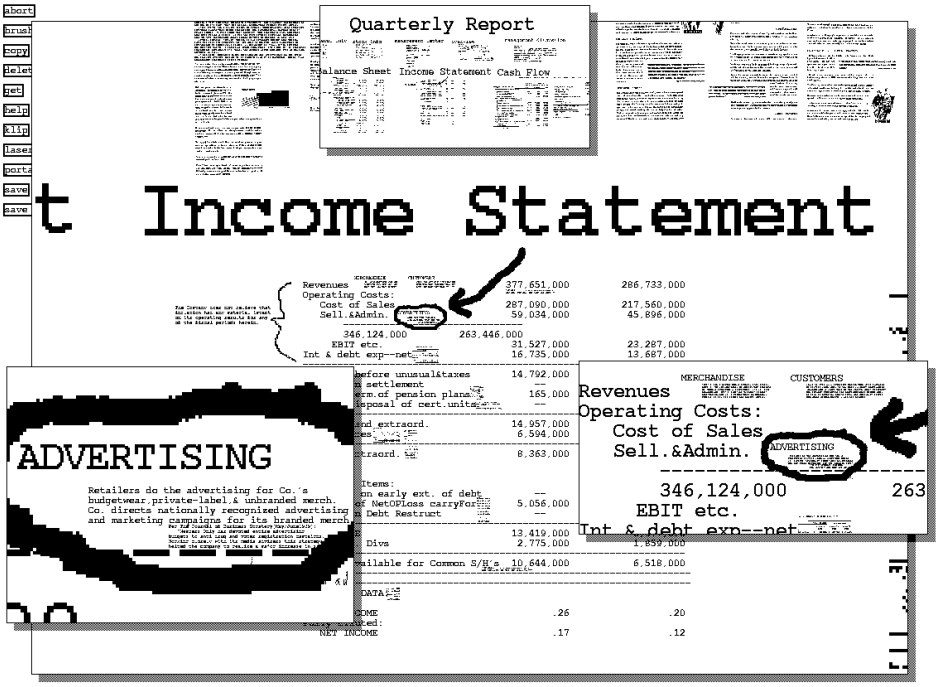



Figura 4.19: Pad [172], una interfaz basada en la técnica *Zoom+Pan*. Se puede ver cómo el usuario puede ir aumentando el detalle con que se muestran ciertas partes del espacio de trabajo. ©1993 ACM

La ventaja principal de este tipo de técnicas, es que aprovechan al máximo el espacio disponible en pantalla, ya que las interacciones permitidas se realizan directamente sobre la información. Pero tienen un inconveniente importante, la pérdida de la visión global sobre la información. Al aumentar la escala de los elementos, aumenta también su tamaño, pero el espacio utilizable sigue siendo el mismo.

**Técnicas Overview+Detail.** Esta familia se caracteriza por dividir el espacio de trabajo en dos zonas bien diferenciadas: la zona de vista global y la zona de vista detallada. La vista global muestra al usuario una visión general de la información con la que debe trabajar; la vista detallada muestra, con mayor grado de detalle, una parte de la información previamente seleccionada en la vista global. Así, ambas vistas están sincronizadas en cuanto a contenidos.

Esta familia de técnicas es de amplio uso en múltiples dominios. Hoy en día, prácticamente todos los sistemas operativos con interfaz gráfica permiten acceder al sistema de ficheros ofreciendo una interfaz *Overview+Detail*; mostramos varios ejemplos en la figura 4.20. También se ha aplicado en herramientas comerciales, como Adobe Acrobat Reader[8], o en herramientas de dominios más específicos como el entorno de desarrollo Eclipse[9] o galerías de imágenes [164].

Las ventajas más destacadas de las técnicas *Overview+Detail* son: permitir navegar por el espacio de trabajo de forma más eficiente [16], facilitar la orientación del usuario dentro del espacio de trabajo [176], y dar al usuario sensación de control [204, 205]. Sin embargo, estas interfaces tienen varias desventajas relacionadas con la separación de las dos vistas: necesitan más espacio en pantalla para ambas, obligan al usuario a integrarlas mentalmente [42, 209], y a cambiar la atención de una a otra

[8]http://www.adobe.com/es/products/acrobat/readstep2.html (2007), Copyright ©2007 Adobe Systems Incorporated. Reservados todos los derechos.

[9]http://www.eclipse.org

Explorer (MS Windows XP) Finder (MacOS X) Konkeror (KDE/Linux)

Figura 4.20: Tres ejemplos de interfaces *Overview+Detail* en herramientas de ayuda al acceso a sistemas de ficheros en interfaces gráficas de usuario

[89].

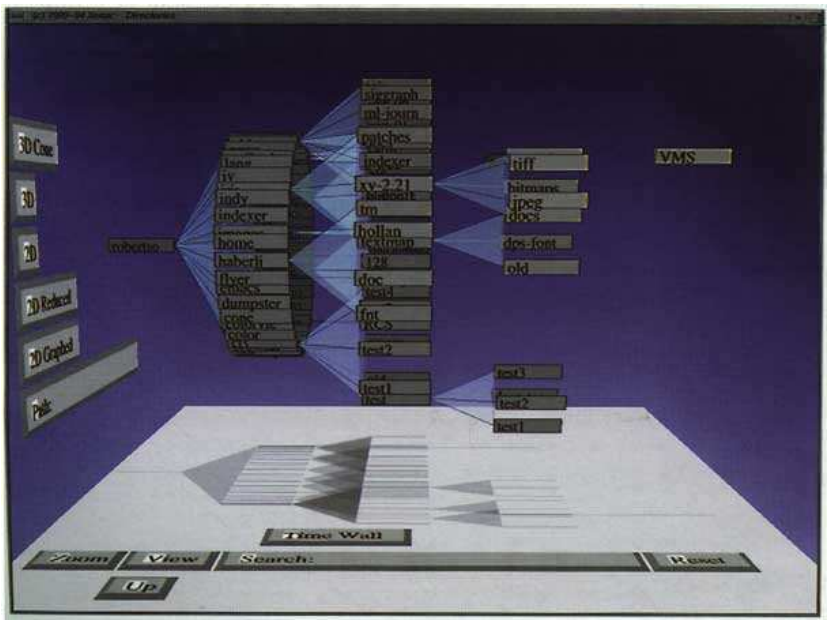
**Técnicas Focus+Context.** Al igual que las técnicas *Overview+Detail*, esta familia muestra simultáneamente vistas globales y detalladas del espacio de trabajo. La diferencia radica en que las técnicas *Focus+Context* integran todo en una sola vista. Así, el espacio de trabajo es ocupado simultáneamente por los elementos de interés (en adelante *foco*), y por el resto de los elementos (en adelante *contexto*). Así, dado que la familia *Overview+Detail* tiene ciertas desventajas en cuanto a la separación de vistas, y que el *contexto* también es importante, se trata de integrar *contexto* y *foco* en una sola vista [42]. El *foco* representa el área de mayor interés para el usuario, y por lo tanto se le dedica mayor cantidad de espacio.

El hecho de clasificar a los elementos en *foco* y *contexto* no significa que sólo existan dos posibles presentaciones de los elementos del espacio de trabajo. Furnas fue uno de los primeros que explicó estas ideas con sus *Logical Fisheye views* [69, 70]. Cada elemento del espacio de trabajo se representa según su grado de interés, que se calcula en función de la importancia intrínseca del elemento, así como la distancia entre el *foco* y el elemento en cuestión.

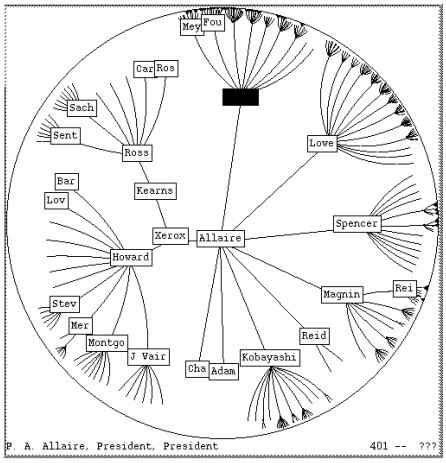
Por otro lado, la reducción del espacio dedicado al *contexto* implica la utilización de técnicas que disminuyan la superficie ocupada por los elementos del *contexto*, permitiendo entre otras, filtrarlos, agregarlos, eliminar partes de ellos mientras que se destacan otras, o distorsionarlos.

La ventaja principal de estas técnicas es la integración de las vistas global y detallada [42]. Sin embargo, Chen [44] llama la atención sobre la pérdida de sensación de continuidad si los cambios de *foco* son demasiado drásticos. Sobre este tema existen estudios como los de Lau et al. [123] y Rensink [183] donde se analiza el efecto de diferentes parámetros de diseño de estas técnicas sobre el reconocimiento de elementos del espacio de trabajo.

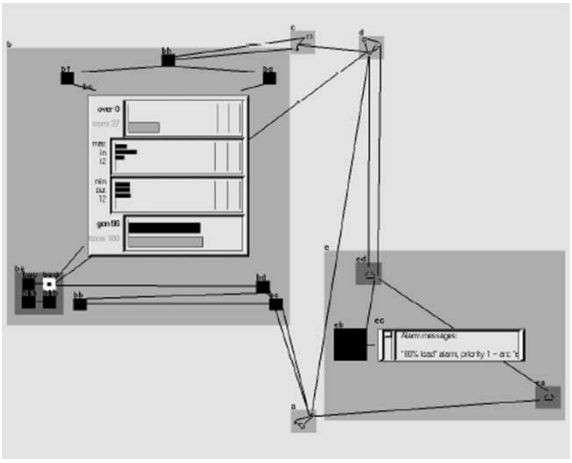
Esta familia de técnicas ha tenido un desarrollo bastante prolijo. En las figuras 4.21, 4.22 y 4.23 mostramos algunos de los ejemplos más representativos.




*ConeTrees* [185] ©1991 ACM

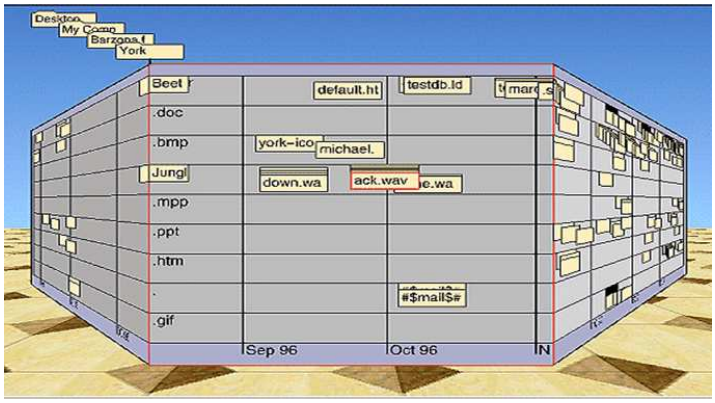



*Hyperbolic Browser* [120] ©1995 ACM

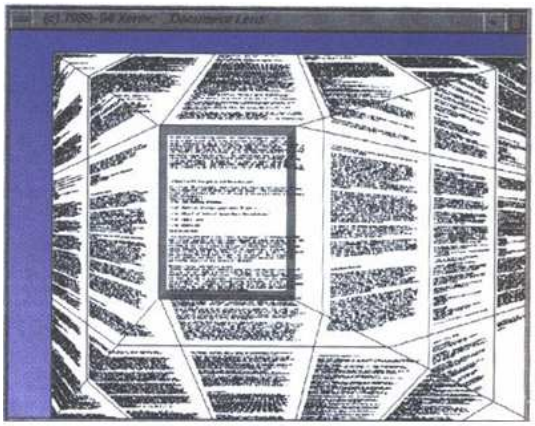


*Continuous Zoom* [12] ©1995 ACM

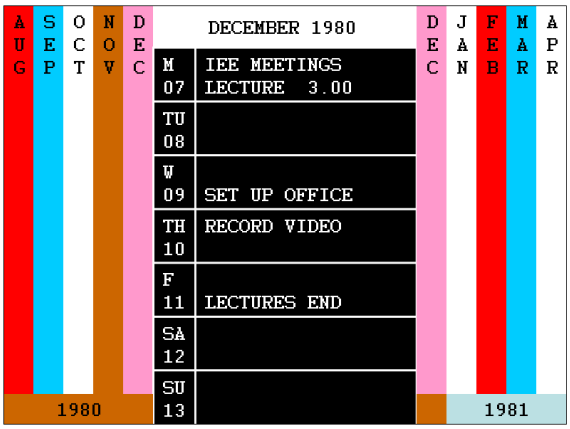
Figura 4.21: Herramientas y técnicas representativas de la familia *Focus+Context*, en este caso centradas en información con estructura jerárquica




*Perspective Wall* [131] ©1991 ACM



*Document Lens* [184] ©1993 ACM

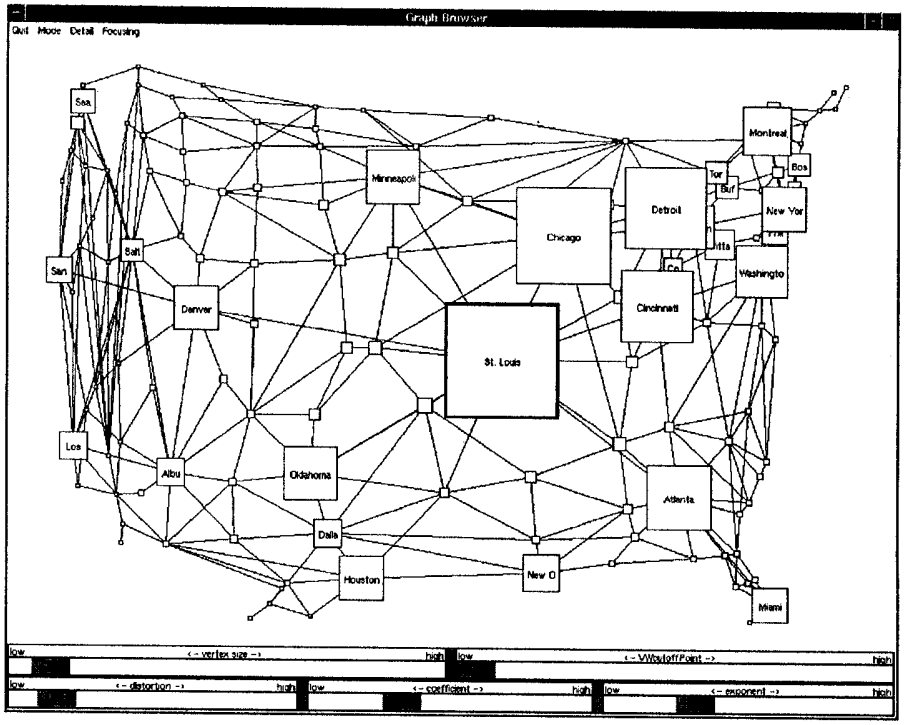



*Bifocal Lens*, adaptado de [210]

```
   1 #define DIG 40
   2 #include <stdio.h>
...4 main()
   5 {
   6       int c, i, x[DIG/4], t[DIG/4], k = DIG/4, noprint = 0;
...8       while((c=getchar()) != EOF){
   9             if(c >= '0' && c <= '9'){
..16             } else {
  17                   switch(c){
  18                         case '+':
..27                       case '-':
..38                       case 'e':
>>39                             for(i=0;i<k;i++) t[i] = x[i];
  40                             break;
  41                         case 'q':
..43                       default:
..46                   }
  47                   if(!noprint){
..57                   }
  58             }
  59             noprint = 0;
  60       }
  61 }
```

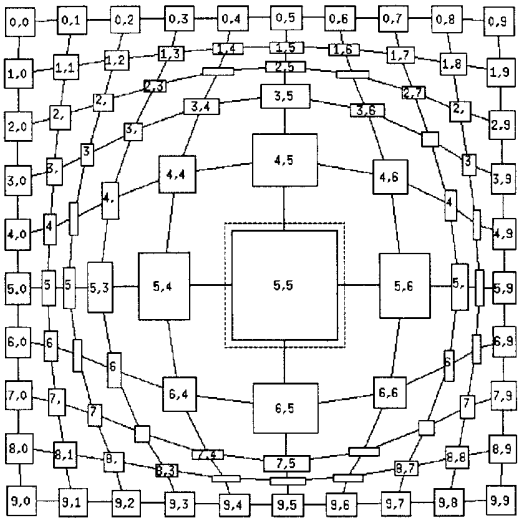
*Logical Fisheye Views* [69] ©ACM 1986

Figura 4.22: Herramientas y técnicas representativas de la familia *Focus+Context*. Continuación de la figura 4.21. *Perspective Wall*, *Document Lens*, y *Bifocal Lens* siguen los principios de las lentes bifocales [210]. Aplicación de *Logical Fisheye Views* [69, 70] a un programa C




*Graphical Fisheye Views* [199] ©ACM 1992

*Flip Zoom* [88] ©ACM 1998



*Rubber Sheet* [200] ©ACM 1993

Figura 4.23: Herramientas y técnicas representativas de la familia *Focus+Context*. Continuación de la figura 4.22. *Graphical Fisheye Views*, basada en las ideas de logical fisheye views (figura 4.22). *Flip Zoom* y *Rubber Sheet*, basadas en las ideas de *Graphical Fisheye Views*

### 4.2.2. Selección de miniaturas con *R-Zoom*

Una vez que tenemos una visión general de las diferentes técnicas de visualización de información que proporcionan vistas globales y detalladas, pasamos a describir *R-Zoom*. En primer lugar describimos los requisitos que hemos identificado para el diseño de la técnica; a continuación, utilizando dichos requisitos, revisaremos las distintas técnicas existentes, detallando finalmente el diseño de la técnica.

#### Análisis de requisitos de *R-Zoom*

Contando con los elementos que debe visualizar esta técnica –representaciones gráficas reducidas de los diferentes estados de ejecución de un programa–, así como con la naturaleza de la tarea a realizar con dichos elementos –selección de las visualizaciones que el usuario crea convenientes para la animación–, hemos identificado tres requisitos de alto nivel (RAN). Por cada uno de ellos, hemos generado los requisitos funcionales (RF) que guiarán el diseño de la técnica:

**RAN1 - orden temporal entre los elementos.** Al ser los elementos visualizaciones de los distintos estadios por los que pasa la ejecución de un programa funcional o algoritmo, el orden en que se generan es parte fundamental del significado de la animación, y por lo tanto debe mantenerse inalterado. Los requisitos funcionales generados son:

**RF1 - poder trabajar con secuencias de elementos.** Nótese que hablamos de secuencias y no de conjuntos o cualquier otra estructura (árboles, grafos cíclicos, ...), ya que la estructura interna existente entre las visualizaciones generadas es lineal.

**RF2 - mantener el orden visual existente entre los elementos.** Para comunicar el orden temporal existente entre las visualizaciones, se utilizará un determinado orden visual que, en la medida de lo posible, deberá permanecer inalterado.

**RAN2 - visión global de los elementos.** Es obvio que para entender la ejecución del programa, se necesita una visión global de la misma. En este caso, el único requisito funcional generado ya venía impuesto anteriormente; la técnica debe:

**RF3 - poder trabajar con versiones reducidas de las visualizaciones.** Para ofrecer una visión global de la ejecución hay que mostrar la mayor cantidad posible de pasos de ejecución, y en términos de representaciones gráficas, eso se traduce en reducir el tamaño de las visualizaciones para poder mostrar un mayor número de ellas simultáneamente.

**RAN3 - visión comprensible de cada uno de los elementos.** Para que el usuario decida si una visualización pasa a formar parte de una animación, deberá comprenderla en su totalidad, teniendo acceso a todo su contenido, y así decidir si ese paso de ejecución es importante como para mostrarse en la animación, o no. Los requisitos funcionales generados son:

**RF4 - mantener la relación de aspecto en las miniaturas.** Tal y como hemos explicado anteriormente en el apartado 4.1.1, la relación de aspecto da información sobre la visualización a quien representa; si se altera, es posible que el usuario no la interprete correctamente, e incluso la confunda con otra miniatura diferente.

**RF5 - permitir al usuario controlar el factor de reducción de las visualizaciones.** Debido a que la comprensión es un concepto muy subjetivo, el factor de reducción aplicado por un

usuario puede ser insuficiente para otro distinto. En consecuencia, si el tamaño de las miniaturas puede aumentar, como el espacio disponible en la pantalla es finito, podría darse el caso de que alguna miniatura no esté siempre visible.

**RF6 - permitir al usuario ver las visualizaciones correspondientes a las miniaturas.** Ya que las miniaturas son heterogéneas en tamaño, forma y contenido, su grado de comprensión puede ser diferente, y el factor de reducción aplicado a una miniatura puede ser demasiado para otra. Así, hemos decidido que el usuario pueda ver a tamaño original, o por lo menos en una versión más detallada, la miniatura seleccionada. Además, de esta forma permitimos un equilibrio entre dos requisitos en parte contrarios: RAN2 y RAN3.

**RF7 - minimizar los cambios en la ubicación de las miniaturas.** Este requisito aparece como consecuencia del anterior (RF6). Como va a ser posible que en un instante dado, una visualización aparezca a tamaño original, y a continuación aparezca en miniatura, la distribución de las miniaturas en el espacio cambiará. Este requisito pretende minimizar el impacto de este cambio en la carga de trabajo del usuario.

A continuación analizamos las técnicas mencionadas en el apartado 4.2.1 según los requisitos funcionales que hemos especificado. Aunque uno de los problemas de la familia de técnicas *Zoom+Pan* es la pérdida del contexto, sí aportarán una característica a *R-Zoom*. Nuestro dominio de aplicación contempla la posibilidad de tener un espacio de trabajo mayor que el área visible en pantalla (RF5), por lo que tendremos que utilizar la técnica de *panning* para poder acceder a todo el espacio de trabajo.

Finalmente, analizamos la familia de técnicas *Focus+Context* según los requisitos anteriormente identificados. Implícitamente, esta familia permite mostrar los elementos del espacio de trabajo a diferentes grados de detalle (RF3 y RF6). Sin embargo, existen muchas técnicas pertenecientes a esta familia que no podemos aplicar debido a los requisitos restantes.

Así, la interfaz que desarrollemos debe trabajar con secuencias de elementos (RF1), por lo que descartamos técnicas tan conocidas como *ConeTrees* [185], *Continuous Zoom* [12], o *Hyperbolic Browser* [120], centradas en la visualización de jerarquías. Existen otras técnicas que permiten trabajan con secuencias como *Bifocal Lens* [210], *Perspective Wall* [131] o *Document Lens* [184], pero no mantienen la relación de aspecto en los elementos del contexto (RF4), por lo que también las descartamos. Finalmente, hemos encontrado dos técnicas que se acercan bastante a los requisitos que hemos especificado: *Flip Zoom* [25, 26, 27] y *Rubber Sheet* [200] (véase figura 4.23). Ambas técnicas permiten trabajar con secuencias de elementos (RF1), así como mantener el orden visual de las mismas (RF2), ya que aseguran que si dos elementos A y B son consecutivos, A siempre aparecerá a la izquierda o por encima de B. También permiten mantener la relación de aspecto en los elementos del contexto (RF4), aspecto prácticamente obligatorio en *Flip Zoom* ([87], p. 40), y más bien secundario en *Rubber Sheet*. Exactamente lo contrario ocurre al minimizar los cambios en la ubicación de las miniaturas (RF7), mientras que *Rubber Sheet* mantiene en la medida de lo posible la localización de los elementos de contexto, *Flip Zoom* da prioridad al foco, siendo este quien delimita el espacio restante disponible y por lo tanto la localización de los elementos del contexto.

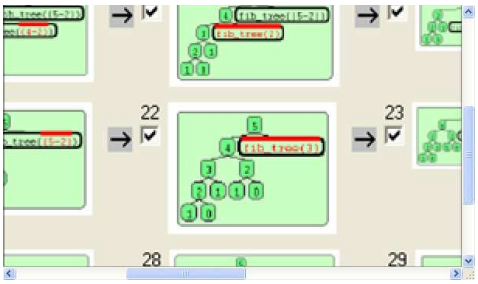
Ninguna de las técnicas que hemos contemplado cubre completamente los requisitos especificados. Por lo tanto, tendremos que diseñar una técnica nueva que integre: facilidades de *panning*, junto con las técnicas más aplicables a nuestra problemática, *Flip Zoom* y *Rubber Sheet*.

### Especificación de R-Zoom

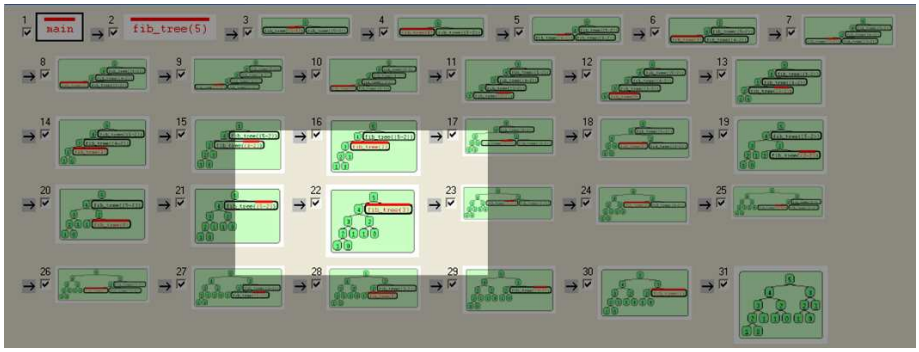
Una vez que hemos identificado las técnicas más afines a nuestro problema pasamos a especificar el diseño de *R-Zoom*. En primer lugar describiremos la representación gráfica de los elementos a visualizar con *R-Zoom*; a continuación su colocación en el espacio de trabajo, y finalmente, el modo en que el usuario podrá interactuar con ellos.

**Representación gráfica de los elementos.** Desde el punto de vista de la técnica *R-Zoom*, los elementos con los que trabaja tienen una representación gráfica intrínseca, las visualizaciones generadas por *WinHIPE*. Dentro de esa representación existen ciertas características ya definidas en la sección 4.1.3, como resaltar el redex de las expresiones funcionales, que ayudan a la consecución del tercer requisito de alto nivel "visión comprensible de cada uno de los elementos". Aun así existen otros puntos que sí entran dentro del ámbito de *R-Zoom*. El tamaño de los elementos del espacio de trabajo no es constante, según los requisitos RF3 y RF6, *R-Zoom* debe poder mostrar los elementos a tamaño original, o reducido, permitiendo al usuario elegir el factor de reducción a aplicar, requisito RF5. Por lo tanto *R-Zoom* generará las reducciones de las visualizaciones de forma dinámica.

Si el usuario puede elegir el factor de reducción, es posible que no se puedan ver simultáneamente todos los elementos, por lo que habrá que optar por una solución de tipo *panning*; en concreto, *R-Zoom* implementa una solución usando la barra de desplazamiento vertical. La razón de esta elección es el orden temporal existente entre los elementos. Se puede saber si un elemento está más arriba o más abajo, reconociendo el estado de ejecución al que representa. Si tuviéramos una interfaz *panning*, con posibilidades de desplazamiento vertical y horizontal, no tendríamos criterio para saber si un elemento está arriba y a la izquierda (véase la figura 4.24), o a la derecha; y como consecuencia los usuarios perderían más tiempo buscando los elementos.



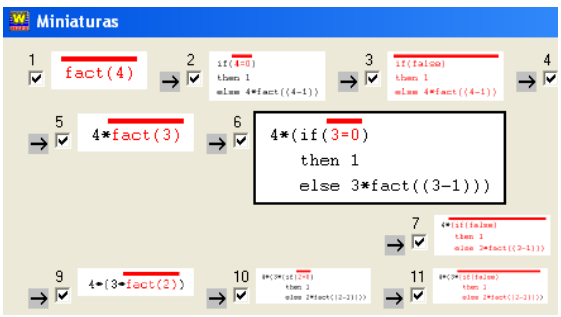

a) Zona visible del espacio de trabajo.



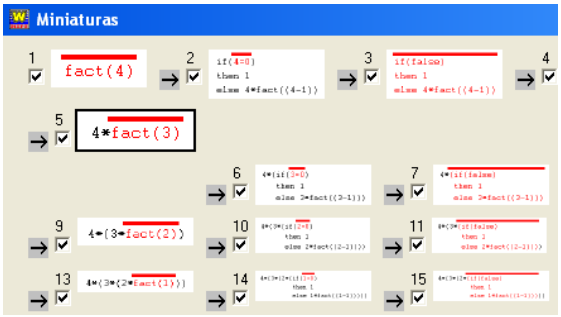

b) Espacio de trabajo completo, la zona sombreada no es visible al usuario.

Figura 4.24: Desplazamiento horizontal y vertical por el espacio de trabajo. Si el usuario quisiera trasladarse a la miniatura 7, sabría que tiene que desplazarse hacia arriba, pero no sabría el sentido horizontal (derecha o izquierda), y tendría que perder tiempo buscando la miniatura

Por otro lado, los elementos son heterogéneos, de distintas formas y tamaños. Puede ocurrir que una visualización sea tan pequeña que no necesite reducción alguna, en este caso la visualización original y su miniatura son idénticas. ¿Cómo sabe el usuario que dicha visualización es el foco de interés? Destacando el foco de interés, no sólo con el aumento de tamaño, que en este caso no serviría, sino con un marco alrededor de ella que la identifica como el *foco* (véase figura 4.25).




El foco es la visualización número 6




El foco es la visualización número 5

Figura 4.25: Resaltado del foco con un marco. Nótese que el tamaño de la visualización 5 es igual en ambos casos, sea el foco o no

**Disposición de los elementos en el espacio de trabajo.** Los requisitos RF1 y RF2 establecen que los elementos tienen entre sí una estructura lineal temporal que no se puede variar. Para mantener dicha estructura, *R-Zoom* ubicará los elementos agrupándolos en filas, de izquierda a derecha, y colocando las filas de arriba a abajo[10] en el espacio de trabajo (véase figura 4.26).

Si todos los elementos del espacio de trabajo son miniaturas, su disposición es la anteriormente descrita. ¿Cómo se colocan los elementos cuando uno de ellos es el foco? Como el foco generalmente ocupa más espacio que las miniaturas, la distribución de todos los elementos cambiará, pero *R-Zoom* debe mantener el orden visual y minimizar los cambios en la ubicación de los elementos, requisitos RF2 y RF7. Para ello hemos tomado tres decisiones en el diseño de *R-Zoom*. Suponiendo que toda miniatura tiene definida su posición según el eje de coordenadas cartesianas:

- Mantener las posiciones horizontales (eje X) de todos los elementos del contexto. Con ello mantenemos el orden visual de los elementos, y por lo tanto el temporal. El efecto es que una miniatura no cambia su posición horizontal mientras no sea seleccionada para ser el foco.

- Minimizar los cambios en la posición vertical (eje Y) de los elementos del contexto. Como la ampliación de tamaño de las miniaturas mantiene la relación de aspecto, la altura del elemento del foco aumentará, por lo tanto, al mantener la posición en el eje X, variará la posición en el eje Y. Restringiremos esta variación a la altura del foco.

- Mostrar el foco lo más cercano a su posición en versión de miniatura. Si es posible el foco se mostrará en el mismo lugar que su miniatura correspondiente, esto evita al usuario cambiar su punto de atención.

A continuación explicamos la colocación de los elementos con el foco en pantalla. R-Zoom divide la fila[11] a la que pertenece la miniatura que va a pasar a ser el foco (miniatura en rojo en las figuras

[10] esta decisión está claramente influida por factores culturales como el orden de lectura occidental.
[11] de aquí el nombre *R-Zoom*, *Row split zoom*.

4.26, 4.27 y 4.28) en dos filas, una con todas las miniaturas anteriores al foco y otra con las miniaturas posteriores (miniaturas en verde y azul respectivamente en las figuras 4.26 y 4.27).

Ahora queda colocar el foco. Siguiendo el principio antes mencionado, trataremos de colocar el foco en las mismas coordenadas que su respectiva miniatura. Para ello tiene que ocurrir que el ancho restante en la primera fila sea mayor o igual que el ancho del foco, como ocurre en la figura 4.26. En este caso la segunda fila se coloca inmediatamente debajo de la primera, la nueva posición de las miniaturas de la segunda fila cambiará en su posición vertical (eje Y), siendo su valor nuevo, el valor antiguo más la altura máxima de las imágenes (miniaturas y foco) de la primera fila más la separación mínima entre filas.

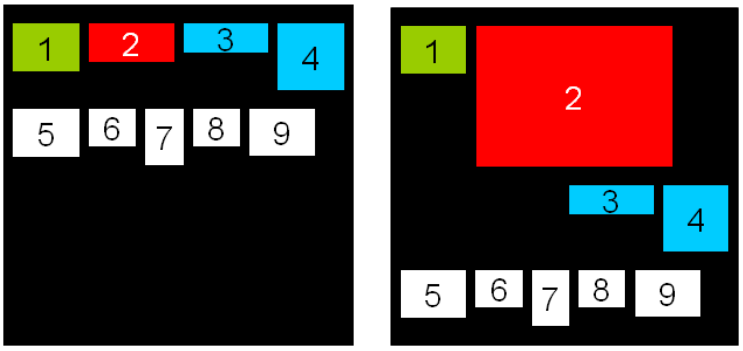
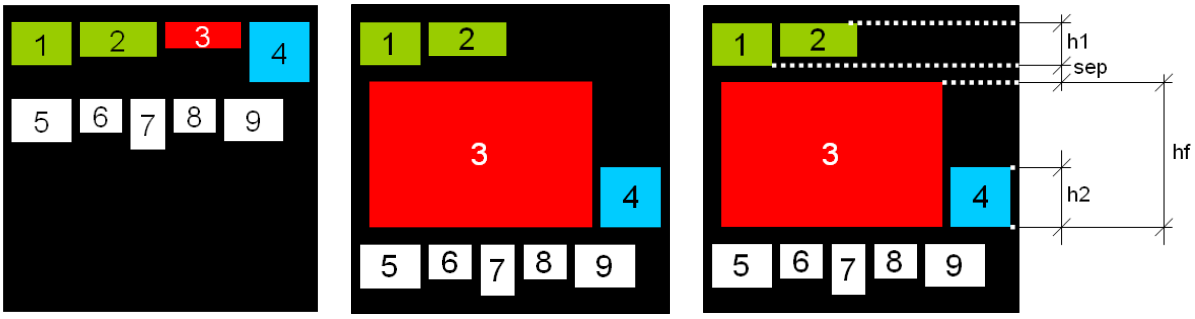



Figura 4.26: Disposición de los elementos con el foco en la primera fila

Si el ancho restante en la primera fila es menor, se prueba en la segunda fila, como ocurre con el elemento número 3 de las figuras 4.27 y 4.28. En este caso la segunda fila también se coloca inmediatamente debajo de la primera, siendo la nueva posición vertical de las miniaturas de la segunda fila el resultado de la siguiente expresión:
$valor\_antiguo + h1 + sep + hf - h2$ ; Si $hf > h2$ (véase figura 4.27)
$valor\_antiguo + sep$ ; Si $hf <= h2$ (véase figura 4.28)
donde *h1* es la altura máxima de las miniaturas de la primera fila, *sep* es la separación mínima entre filas, *hf* es la altura del foco y *h2* es la altura máxima de las miniaturas de la segunda fila.




a) b) c)

Figura 4.27: Disposición de los elementos con el foco en la segunda fila

Finalmente, si el foco es más ancho que cualquiera de los dos huecos de las filas, se elige el hueco más grande, y se escala el foco a ese tamaño. Si el usuario ha elegido un elemento como foco, es que desea verlo con mayor detalle; nosotros hemos decidido ofrecerle el mayor detalle posible dentro de las decisiones anteriormente mencionadas.

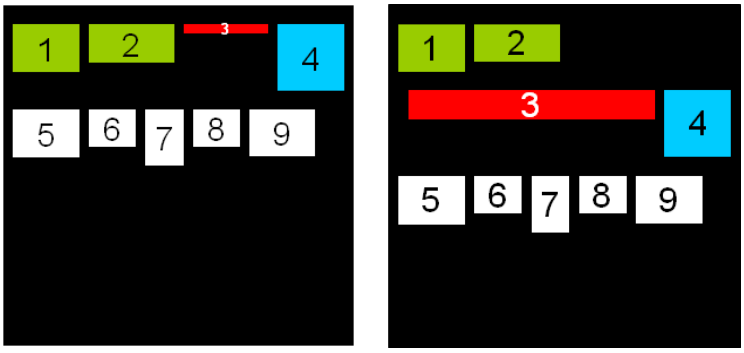



Figura 4.28: Disposición de los elementos con el foco en la segunda fila, cuando el foco es menos alto que el resto de miniaturas de la segunda fila

Las figuras 4.29 y 4.30 muestran dos situaciones reales donde el foco se encuentra en la primera y segunda fila respectivamente.

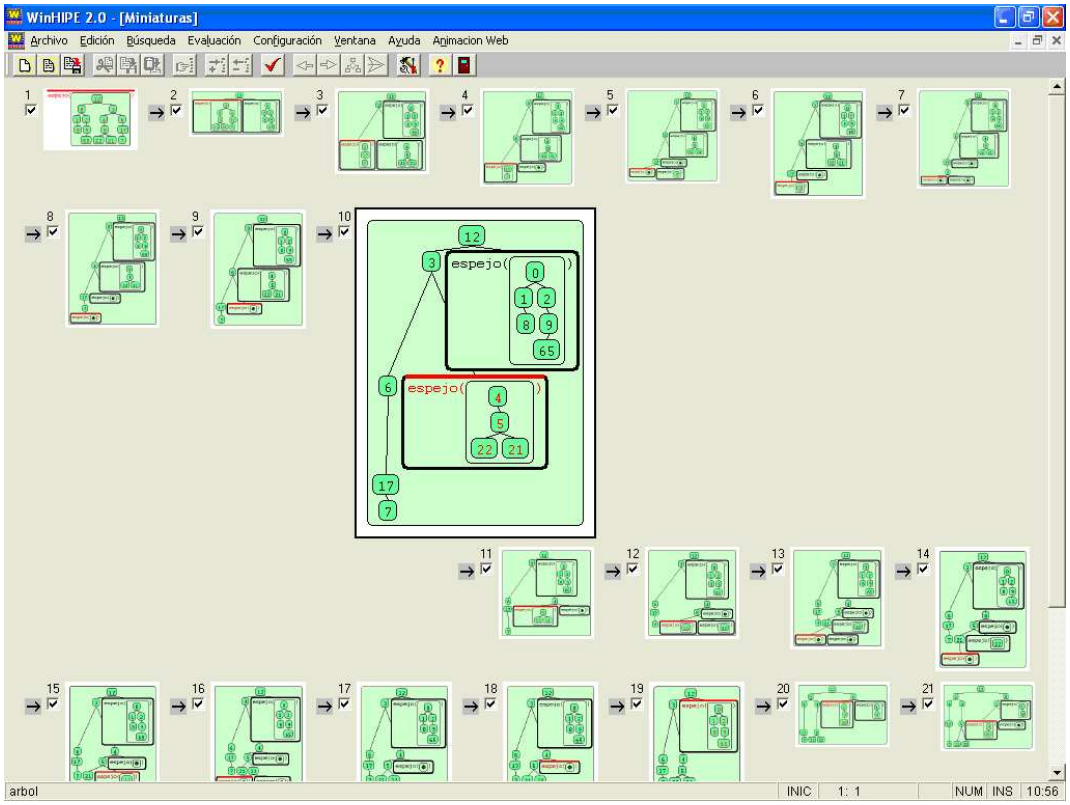



Figura 4.29: Disposición de los elementos con el foco en la primera fila

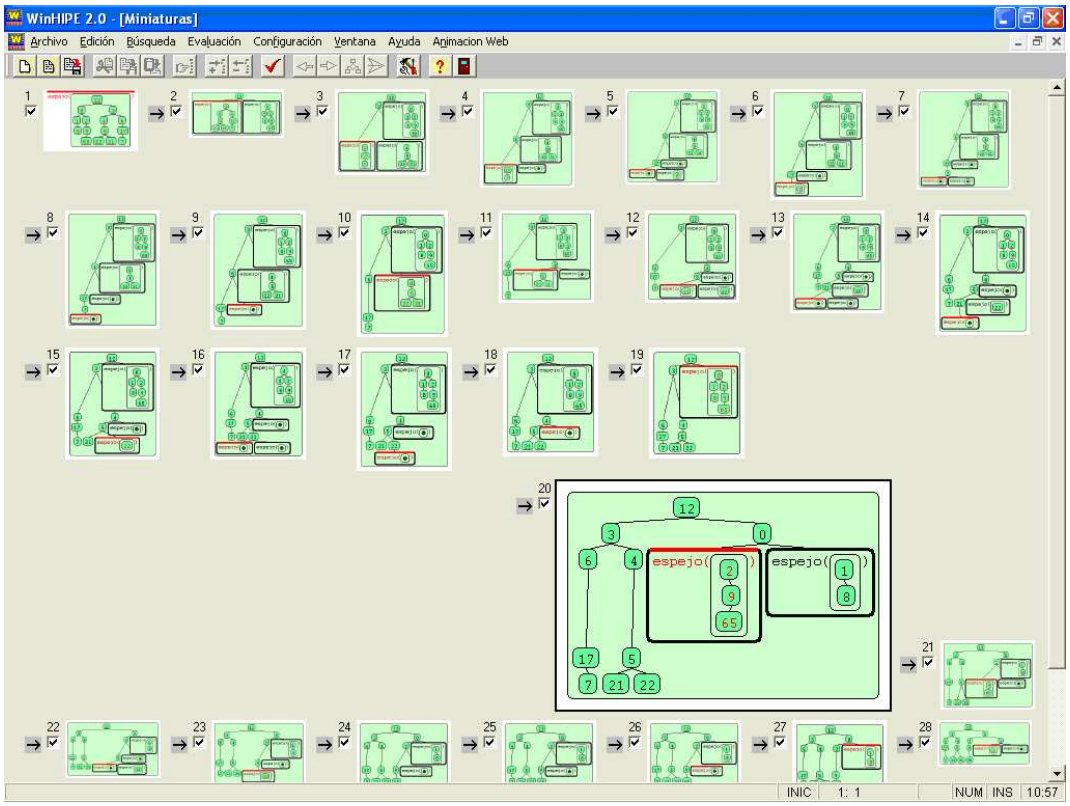



Figura 4.30: Disposición de los elementos con el foco en la segunda fila

**Interacción con los elementos y el espacio de trabajo.** El funcionamiento básico de R-Zoom se divide en dos estados: *no-focus*, cuando en la ventana no hay foco seleccionado, y por lo tanto todos los elementos existentes son miniaturas; y *focus*, cuando sí se está mostrando una imagen como foco. A continuación explicamos cuándo se producen las transiciones y cómo afectan las distintas posibilidades de interacción existentes. También mostramos un dibujo esquemático en la figura 4.31.

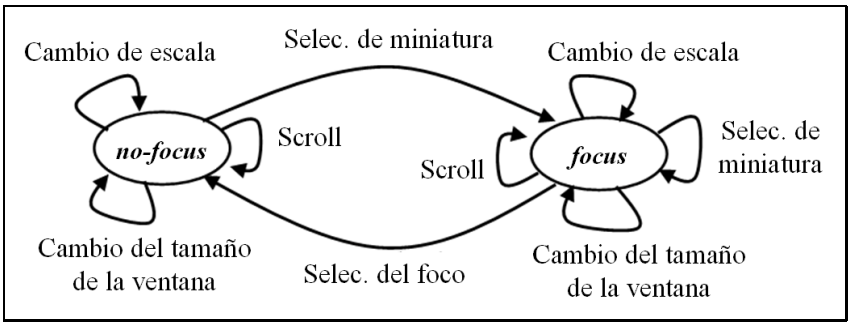



Figura 4.31: Transiciones entre los estados de R-Zoom

Por defecto, cuando el usuario comienza a trabajar con una animación, la interfaz se encuentra en el estado *no-focus*. Transitamos al estado *focus* cuando el usuario selecciona una miniatura para su visión detallada. Si estando en el modo *focus*, se selecciona otra miniatura, se ejecutarán consecutivamente el paso al modo *no-focus*, y de nuevo al modo *focus* con la nueva miniatura en su versión detallada. Si estando en el modo *foco*, se selecciona el propio foco, se pasa al modo *no-focus*.

El resto de interacciones posibles son: desplazamiento vertical, cambio de escala aplicada a las

miniaturas, y cambio del tamaño de la pantalla. El resultado del uso del *scroll* es evidente: la parte visible del espacio de trabajo cambia, sin afectar al estado de los elementos (el foco sigue siendo el foco y las miniaturas siguen siendo miniaturas). Cuando el usuario cambia el factor de escala aplicado a las miniaturas, redistribuimos de nuevo todos los elementos en el espacio de trabajo siguiendo las directrices de disposición de los elementos en el espacio de trabajo explicadas en el apartado anterior. Finalmente, si se produce un cambio en el tamaño de la ventana hay que distinguir entre varios casos:

**Cambia la altura de la ventana.** En este caso estamos cambiando el área visible del *scroll*, pero la disposición de los elementos en el espacio de trabajo no cambia.

**Cambia la anchura de la ventana.** *R-Zoom* distribuye los elementos en filas colocando tantos elementos como sea posible. Si cambiamos el ancho de la ventana, estaremos cambiando el espacio disponible de las filas, y por lo tanto se podrá producir un cambio en la disposición de los elementos. Sin embargo, no siempre se produce dicho cambio. Para optimizar el rendimiento de *R-Zoom* guardamos las anchuras máxima y mínima que no producirían ningún cambio en la disposición actual de los elementos.

### 4.2.3. Evaluación de *R-Zoom*

Nos hemos centrado en evaluar la usabilidad de la técnica de visualización de información diseñada. Siguiendo el estándar de usabilidad proporcionado por la norma ISO 9241-11 [98], analizaremos la eficacia, eficiencia y la satisfacción de los usuarios al realizar las tareas con ambas interfaces. Como el objetivo principal de esta técnica es seleccionar las visualizaciones que formarán parte de la animación, las tareas de la evaluación consistirán en la búsqueda de determinadas visualizaciones dentro de una colección de miniaturas.

Somos conscientes de las ventajas de las interfaces *Overview+Detail*, así que en esta evaluación compararemos *R-Zoom* con una solución *Overview+Detail*, en adelante *O+D*, a la que hemos añadido las mejoras relacionadas con la generación de las miniaturas descrita en la sección 4.1(véase figura 4.32).

#### Diseño de la evaluación

Esta evaluación consistió en una sesión experimental de dos horas de duración. A continuación detallamos la infraestructura, los participantes y el protocolo de la sesión.

**Infraestructura.** La infraestructura consistió en ordenadores tipo *PC* con el sistema operativo *MicroSoft Windows XP Professional* y la siguiente configuración hardware: procesador Pentium III 933 MHz, 256MB de memoria RAM, monitor de 17" Hitachi CM620ET funcionando a una resolución de 1024x768 píxeles y tarjetas gráficas Intel 82815 AGP 32MB. Hemos utilizado dos aplicaciones[12] para realizar la evaluación: el software de monitorización que mide tanto el tiempo empleado por los usuarios en realizar las tareas como los errores cometidos por estos, y la aplicación *WinHIPE* especialmente configurada para la utilización de las interfaces a comparar, *R-Zoom* y *O+D*. Configuramos ambas interfaces para que usaran todo el espacio disponible en la pantalla, así como aplicar un factor de reducción inicial a todas las visualizaciones de un 50 %. Este factor de reducción representa una solución de compromiso entre el número de miniaturas visibles simultáneamente en pantalla y su grado de

[12] Ambas desarrolladas en Borland Delphi 5


Figura 4.32: Versión de la interfaz O+D para la selección de miniaturas

comprensión. Rensink [183] demuestra que los factores de reducción menores del 50 % no tienen efecto en tareas de reconocimiento de formas y siluetas, característica fundamental en el reconocimiento de las visualizaciones generadas en *WinHIPE*.

**Participantes.** Los usuarios que participaron en la evaluación eran usuarios finales de las interfaces a evaluar, con visión normal o corregida, y cuya participación era voluntaria sin ningún tipo de incentivo. Dichos usuarios eran estudiantes de primer año de las titulaciones Ingeniería Técnica en Informática de Sistemas y de Gestión, en la Universidad Rey Juan Carlos. Los estudiantes, un total de 43, estaban divididos en dos turnos distintos, uno para cada titulación. Para tratar de evitar posibles desequilibrios, creamos dos grupos –A y B– mezclando estudiantes de las dos titulaciones. La asignación de los estudiantes a los grupos fue aleatoria y los porcentajes de estudiantes de ambas titulaciones estaban equilibrados en los dos grupos. En total, el grupo A tenía 26 estudiantes y el grupo B 17; ambos grupos usaron las dos interfaces, pero en distinto orden.

**Protocolo.** Las tareas a realizar durante la evaluación incluyeron ampliar las miniaturas así como navegar a través de las colecciones de estas buscando una en particular. Las visualizaciones a buscar en cada tarea son pistas en sí mismas, ya que al representar estados de ejecución de un algoritmo, el usuario puede intuir la posición relativa de la miniatura en la colección. Consideramos que una tarea se termina con éxito si el usuario introduce el número correcto asociado a la visualización objetivo en la caja de texto del software de monitorización y hace click en el botón "siguiente" (véase la figura 4.33). Inmediatamente después se le presenta al usuario una nueva tarea.

Ambos grupos utilizaron las dos interfaces pero en distinto orden. Así disponemos de datos para comparar medidas entre usuarios con las dos interfaces y entre interfaces con el mismo usuario. El orden de ejecución de las tareas, del que mostramos un esquema en la figura 4.34, fue el siguiente. En primer lugar el instructor encargado de la evaluación explicó a cada grupo cómo se manejaba

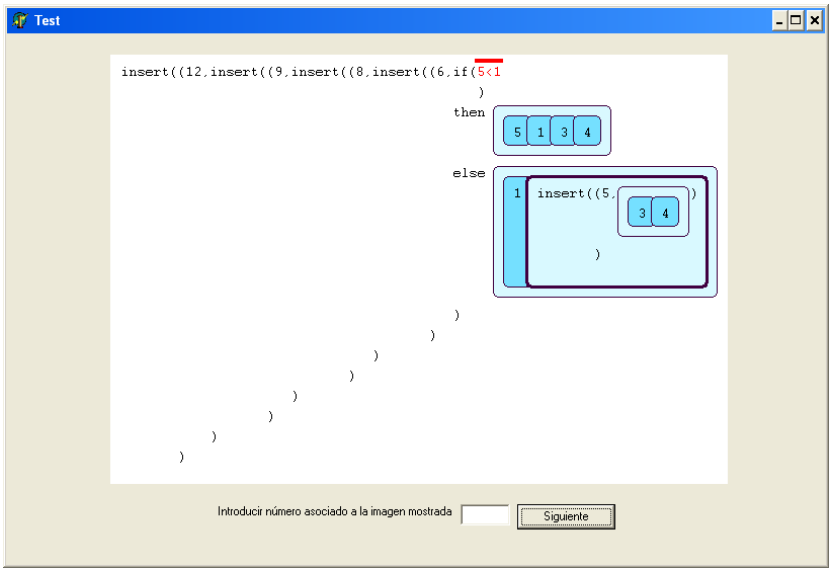



Figura 4.33: Figura captura del software de monitorización

cada interfaz, *R-Zoom* al grupo A y *O+D* al grupo B. A continuación, cada grupo realizó una tarea de entrenamiento con la interfaz correspondiente; esta tarea era de las mismas características que aquellas a monitorizar. Las colecciones de miniaturas utilizadas tanto para las tareas de entrenamiento como para las monitorizadas eran resultado de la ejecución de diferentes algoritmos.

Los usuarios realizaron un total de nueve tareas con cada interfaz: tres tareas diferentes (con el objetivo de búsqueda al principio, a la mitad y al final de la colección) con tres tipos diferentes de colecciones (ocupando una pantalla, pantalla y media, y tres pantallas). La figura 4.34 muestra un esquema del protocolo seguido en el experimento. El orden de presentación de las tareas estaba fijado previamente y los usuarios no sabían que las visualizaciones objetivos estaban en las zonas antes mencionadas. La colección y la imagen objetivo asociadas a cada tarea eran las mismas, independientemente del grupo al que pertenecía el usuario. Para evitar que los usuarios se quedaran atascados en una tarea, se les permitió un límite de 4 fallos por tarea; después del cuarto error, el software de monitorización comunicaba al usuario la respuesta correcta y continuaba con la siguiente tarea.

Los usuarios del grupo A realizaron primero nueve tareas con la interfaz *R-Zoom*. Simultáneamente, los usuarios del grupo B realizaron las mismas tareas con la interfaz *O+D*. Una vez completadas las primeras nueve tareas, cada grupo cambió de interfaz y realizó otras nueve tareas. Estas tareas comprendían diferentes imágenes, en las mismas posiciones –principio, mitad y final de la colección–, pero en diferente orden a las anteriores y con las mismas colecciones anteriormente usadas.[13]

Las variables dependientes de la evaluación fueron: el número de errores por cada tarea, el tiempo empleado en realizar las tareas y la satisfacción de los usuarios con las interfaces. El tiempo empleado en realizar la tarea era el tiempo transcurrido desde que el usuario hizo clic por última vez en el botón "siguiente" (comienzo de la primera tarea, o fin de la tarea anterior), hasta que el usuario hace clic en el botón "siguiente" con el número correcto asociado a la visualización objetivo.

Una vez terminadas todas las tareas, pedimos a los usuarios que hicieran comentarios sobre las interfaces evaluados. Así medimos la satisfacción de los usuarios. Para ello les pedimos que contestaran

[13]En la dirección web `http://www.escet.urjc.es/~jurquiza/Tesis/rzoom.html` están disponibles un conjunto de vídeos de demostración que sirven como ejemplo a una sesión de la evaluación.

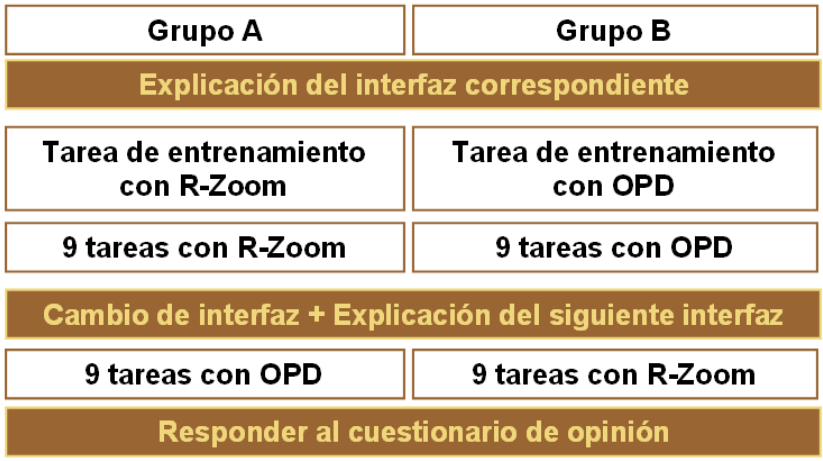



Figura 4.34: Esquema del protocolo seguido en la evaluación

a un cuestionario con preguntas sobre: impresión general, utilidad, facilidad de uso, necesidad de estas interfaces en la construcción de animaciones de programas, ventajas, e inconvenientes. En el apéndice A.1 se puede encontrar una copia de dicho cuestionario.

### Resultados de la evaluación

Hemos analizado los resultados de la evaluación de dos formas distintas: basado en grupo-interfaz, y basado en tarea. El análisis basado en grupo-interfaz (en adelante GIB) referencia los datos según el grupo (A o B) y la interfaz usada, RZ (*R-Zoom*) u OD (*O+D*). Así podremos diferenciar cuatro grupos distintos: ARZ, AOD, BRZ y BOD. Además, este análisis contempla la experiencia de los usuarios con las tareas de la evaluación. El grupo A comenzó con la interfaz *R-Zoom*, por lo que ARZ implica sólo experiencia de una tarea, la de entrenamiento. Lo mismo ocurre con el BOD. AOD y BRZ implican una experiencia de la tarea de entrenamiento junto con las nueve tareas con la primera interfaz utilizada, *R-Zoom* en el caso de AOD y *O+D* en el caso de BRZ.

En el análisis basado en tarea (en adelante TB), referenciamos los datos usando el análisis GIB junto con la información de la colección y la imagen objetivo. Así BOD12 identifica a la tarea donde los usuarios del grupo B, usando la interfaz *O+D*, deben encontrar la primera[14] imagen de la segunda colección.

A continuación verificamos la validez de los datos medidos en cuanto a errores cometidos y tiempo empleado por los usuarios, para finalmente analizar de forma separada los resultados de eficacia, eficiencia, y satisfacción de los usuarios.

**Validez de los datos para el estudio comparativo.** Idealmente, los usuarios del mismo grupo habrían realizado las tareas bajo las mismas circunstancias; de esta forma, podemos analizar los datos independientemente de los diferentes turnos en los que se dividían los grupos. Hemos comprobado si los diferentes turnos pertenecientes al mismo grupo realmente pertenecen a la misma población. Los datos referentes a los errores cometidos no muestran diferencia alguna entre turnos de un mismo grupo, por lo que podrán ser tratados como dos grupos en el estudio comparativo. Sin embargo, sí hemos encontrado

[14]Denota orden de exposición de la tarea, no localización de la visualización en la colección

diferencias en los datos sobre el tiempo empleado por los usuarios para realizar las siguientes tareas: AOD12, BOD22, BRZ11 y BRZ12. Por lo tanto, los datos sobre estas tareas no se utilizarán para el análisis comparativo, ya que los distintos turnos de cada grupo no se pueden tratar como uno solo. Los detalles del análisis estadístico están disponibles en el apéndice A.2.

Además detectamos otra anomalía en las tareas AOD11 y BRZ11, la primera tarea realizada después del cambio de interfaz en cada grupo. El tiempo invertido por los usuarios en esta tarea, era aproximadamente diez veces el tiempo invertido en las otras dos tareas de la misma colección (véase la tabla 4.5). Este dato nos dice que deberíamos haber incluido una tarea de entrenamiento en el cambio de interfaz.

| | xxx11 | xxx12 | xxx13 |
|---|---|---|---|
| AOD | 482.61 s | 34.11 s | 42.88 s |
| BRZ | 356.94 s | 33.88 s | 34.88 s |

Tabla 4.5: Comparación de tiempos de la primera tarea después del cambio de interfaz (AOD11 y BRZ11) con las dos tareas siguientes en cada grupo

En conclusión, para el resto del análisis comparativo, decidimos ignorar los datos de tiempo referentes a las tareas involucradas: AOD11, AOD12, BOD22, BRZ11 y BRZ12. Pero existen más implicaciones, el análisis GIB estudia los datos independientemente de la tarea en concreto; por lo tanto al realizar las comparaciones entre los cuatro grupos habrá que comprobar la influencia de haber ignorado los datos de las tareas anteriormente referidas. Hemos descubierto que el tiempo empleado por los usuarios para completar las tareas y, tanto la colección como la imagen involucrada en dichas tareas, no son independientes (véase figura A.5 del apéndice A.2). Por lo tanto, si ignoramos las tareas AOD11 y AOD12, todas las comparaciones con AOD deben ignorar sus respectivas tareas XXX11 y XXX12, lo mismo pasará con BOD22, BRZ11 y BRZ12.

**Análisis comparativo de la eficacia.** Hemos caracterizado la eficacia de las interfaces como el número de errores cometidos por los usuarios al realizar las tareas. De las 774 tareas realizadas (43 usuarios x 2 interfaces x 3 colecciones x 3 visualizaciones), sólo detectamos errores en 59: 55 con un error y 4 con dos errores. A primera vista se puede decir que ambas interfaces son bastante eficaces.

Tras realizar el análisis GIB, encontramos diferencias significativas entre algunas de las parejas *grupo-interfaz*. La tabla 4.6 muestra el resumen de los datos sobre errores cometidos. Así, para cada pareja grupo-interfaz proporcionamos el número total de errores cometidos, independientemente de si son en una misma tarea o en diferentes; el promedio de errores por tarea, que es la división del número de errores cometidos entre el número de tareas realizadas; y la tasa de error, que es el número de errores cometidos dividido entre el número máximo de errores que podrían haberse cometido (máximo de cuatro errores por tarea). En la tabla 4.7 mostramos el análisis de diferencias significativas.

El análisis de diferencias significativas muestra que añadiendo experiencia al uso de R-Zoom (ARZ-BRZ) ayuda a los usuarios a cometer un 72,0 % menos de errores, cosa que no pasa con la interfaz O+D (AOD-BOD). Por otro lado, entre usuarios con la misma experiencia (BRZ-AOD), aquellos que usaron R-Zoom cometieron un 70,8 % menos de errores. Finalmente, la comparativa de usuarios poco experimentados no ha dado diferencia alguna.

También hemos encontrado diferencias significativas en el análisis TB (véase el apéndice A.3.2). Sin embargo estas tienen un carácter más bien anecdótico ya que, de 41 comparaciones posibles, sólo

| | Número de errores | Promedio de errores por tarea | Tasa de error |
|---|---|---|---|
| BOD | 13 | 0.085 | 0.021 |
| ARZ | 24 | 0.102 | 0.025 |
| BRZ | 4 | 0.026 | 0.007 |
| AOD | 22 | 0.094 | 0.024 |

Tabla 4.6: Medidas GIB de la eficacia

| | AOD | **BRZ** | ARZ |
|---|---|---|---|
| BOD | $U = 5910{,}5, p = 0{,}251$ | $p = 0{,}039$ | $U = 5846{,}5, p = 0{,}168$ |
| ARZ | $p = 0{,}770$ | $U = 16607{,}5, p = 0{,}006$ | |
| **BRZ** | $U = 16835{,}0, p = 0{,}018$ | | |

Tabla 4.7: Análisis GIB de diferencias significativas en la eficacia (los detalles están disponibles en el apéndice A.3.1)

dos han dado resultados significativos y no podemos sacar conclusiones de ellas.

**Análisis comparativo de la eficiencia.** Para caracterizar la eficiencia de las interfaces hemos medido el tiempo empleado en realizar cada tarea. Aunque las tasas de error son bastante bajas, han provocado diferencias entre los diferentes grupos, por lo que la eficiencia tendrá en cuenta tanto la eficacia (errores cometidos) como los recursos utilizados (tiempo empleado). Hemos caracterizado la eficiencia con la siguiente fórmula: $tiempo/(4 - errores)$. Cuanto menor sea este valor, mejor será el resultado de la eficiencia.

De las seis posibles comparaciones a realizar con el análisis GIB, detectamos tres con diferencias significativas. En la tabla 4.8 mostramos un resumen de los resultados en términos de eficiencia y tiempo empleado por los usuarios en realizar las tareas.

| | Eficiencia (media , desviación típica) | Tiempo (media , desviación típica) |
|---|---|---|
| BOD | $M = 27,41\ SD = 18,680$ | $M = 106,18\ SD = 71,148$ |
| ARZ | $M = 24,79\ SD = 21,609$ | $M = 94,31\ SD = 78,241$ |
| BRZ | $M = 17,95\ SD = 14,712$ | $M = 70,93\ SD = 57,087$ |
| AOD | $M = 22,26\ SD = 16,012$ | $M = 85,39\ SD = 60,497$ |

Tabla 4.8: Medida GIB de la eficiencia. Nótese que estas medidas son aproximadas, ya que los datos que se tendrán en cuenta en el estudio comparativo dependen de si los grupos que comparamos tienen alguna tarea anómala o no

De nuevo, el análisis GIB nos muestra que todas las comparaciones con el grupo BRZ, usuarios experimentados con la interfaz de *R-Zoom*, dan diferencias significativas favorables a dicho grupo (véase la tabla 4.9). Como podemos ver, la experiencia afecta a la eficiencia con la interfaz *R-Zoom*, comparación de ARZ y BRZ, mejorando los resultados en un 21,58 %; mientras que en la interfaz *O+D* esto no ocurre. Al comparar grupos de usuarios con la misma experiencia; si esta es poca (ARZ-BOD) no detectamos diferencia alguna, pero si los usuarios son experimentados (AOD-BRZ), aquellos que usaron *R-Zoom* lo hicieron más eficientemente, con una mejora del 22,64 % sobre los que utilizaron

*O+D*. Los otros dos resultados (BRZ-BOD y ARZ-AOD) son consecuencia de los anteriores.

| | AOD | **BRZ** | ARZ |
|---|---|---|---|
| BOD | $U = 7409{,}5, p = 0,351$ | $p = 0{,}000$ | $U = 12867{,}0, p = 0{,}157$ |
| ARZ | $p = 0{,}430$ | $U = 8810{,}5, p = 0{,}006$ | |
| **BRZ** | $U = 8956{,}5, p = 0{,}011$ | | |

Tabla 4.9: Análisis GIB de diferencias significativas en cuanto a tiempo empleado (los detalles están disponibles en el apéndice A.4.1)

Hemos aplicado el análisis TB a los resultados del análisis previo GIB, así que solamente hemos comparado las tareas correspondientes a las siguientes parejas: BOD-BRZ, ARZ-BRZ y BRZ-AOD. Los detalles estadísticos de este estudio están disponibles en el apéndice A.4.2. De las 20 comparaciones realizadas (3 parejas con 9 tareas cada una, menos 7 anómalas), hemos detectado 10 diferencias significativas, siempre en favor de los usuarios experimentados utilizando *R-Zoom*, que corresponden al grupo BRZ. Las tablas 4.10 a), b) y c), muestran los resultados de estas 10 comparaciones.

| Tareas BOD vs BRZ | Resultado del análisis | % de mejora |
|---|---|---|
| 21 | t(16)=3,107 , p=0,007 | 28,07 % |
| 23 | t(16)=3,354 , p=0,004 | 40,98 % |
| 31 | t(16)=2,157 , p=0,047 | 22,20 % |
| 32 | t(16)=2,543 , p=0,022 | 39,56 % |

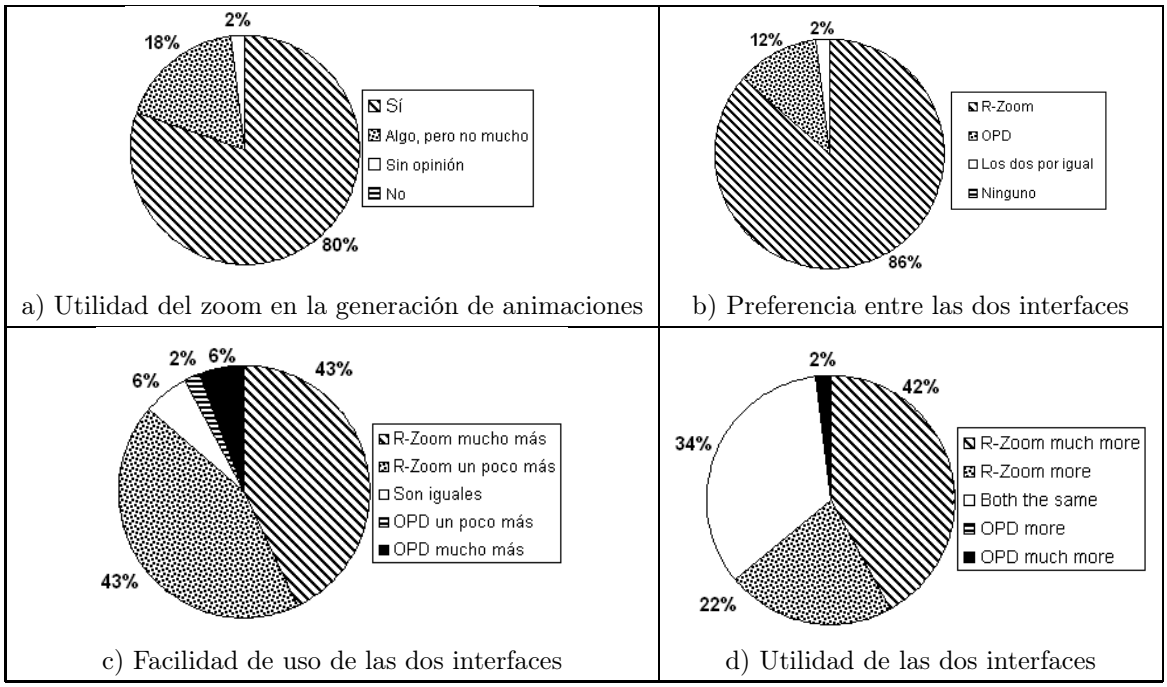
| Tareas ARZ vs BRZ | Resultado del análisis | % de mejora |
|---|---|---|
| 21 | t(41)=2,830 , p=0,007 | 28,17 % |
| 22 | U=114,000 , p=0,008 | 44,01 % |
| 23 | t(41)=2,276 , p=0,028 | 33,11 % |
| 32 | t(41)=2,322 , p=0,025 | 26,94 % |

| Tareas AOD vs BRZ | Resultado del análisis | % de mejora |
|---|---|---|
| 22 | t(38,012)=2,312 , p=0,026 | 34,04 % |
| 23 | t(41)=2,592 , p=0,013 | 34,67 % |

Tabla 4.10: Análisis TB de eficiencia de las parejas BOD vs BRZ, ARZ vs BRZ, y AOD vs BRZ

Con los resultados anteriores podemos ver cómo la mejora media en las tareas relacionadas con la segunda y tercera imagen, 36,13 % y 36,25 % respectivamente es mayor que la mejora media de la primera imagen, 26,15 %. Además la mayoría de las tareas relacionadas con la segunda visualización, 4 tareas de 5 posibles, han dado diferencia significativa. Las razones para estos resultados se pueden justificar con una de las características principales de *R-Zoom*, mantener en la medida de lo posible la localización de los elementos del contexto. Así, después de haber encontrado la primera visualización, el usuario ha tenido la oportunidad de ver los elementos del contexto, donde se encuentran la segunda y tercera visualización, o al menos otras que actúan como pistas o indicadores de dónde se pueden encontrar.

**Análisis comparativo de la satisfacción de los usuarios.** Para medir la satisfacción de los usuarios con ambas interfaces usamos un cuestionario (véase apéndice A.1). En primer lugar les preguntamos sobre la necesidad de interfaces con posibilidad de zoom para construir animaciones de programas o algoritmos. El 98 % de los usuarios están de acuerdo en mayor o menor medida sobre la necesidad de estas utilidades de zoom, véase la figura 4.35a. En cuanto a la satisfacción de los usuarios por una u otra interfaz, figura 4.35b, el 86 % de los usuarios prefieren *R-Zoom* frente al 12 % que prefieren la interfaz *O+D*. A continuación preguntamos a los usuarios sobre la facilidad de uso, figura 4.35c, el 86 % de los usuarios piensan que *R-Zoom* es más o mucho más fácil de usar que *O+D*, mientras que sólo el 8 % piensan lo contrario. A continuación, preguntamos a los usuarios por la utilidad de ambas interfaces, figura 4.35d, el 64 % de los usuarios piensan que *R-Zoom* es más o mucho más útil que *O+D* y el 34 % piensan que ambas interfaces son igual de útiles.




a) Utilidad del zoom en la generación de animaciones

b) Preferencia entre las dos interfaces

c) Facilidad de uso de las dos interfaces

d) Utilidad de las dos interfaces

Figura 4.35: Opinión de los usuarios

Finalmente, ofrecimos a los estudiantes la posibilidad de comentar aspectos positivos y negativos de ambas interfaces. El detalle de los comentarios hechos por los usuarios referentes a estos aspectos está disponible en el apéndice A.5. Los comentarios se centraron principalmente en tres aspectos: la interfaz ofrece una visión global de la animación, separación/integración de las vistas global y detallada de las visualizaciones, y sobre la orientación en la colección de miniaturas; nosotros hablaremos de los más relevantes, que han sido mencionados por al menos un 20 % de los usuarios. El 40 % de los usuarios identificó como una ventaja de la interfaz *R-Zoom* que ofreciera una *visión global* de la colección de miniaturas. Además, el 24 % mencionaron como otra ventaja de *R-Zoom* la *integración de las vistas* global y detallada; la gran mayoría de estos usuarios, un 20 % del total, identificaron a su vez como inconveniente el hecho de que la interfaz *O+D separe las vistas* global y detallada.

## 4.3. Evaluación del proceso de construcción

Hemos evaluado con éxito los dos enfoques elegidos para tratar el problema de las miniaturas comprensibles: aumentar la calidad de las miniaturas y diseñar una técnica que permita la integración de minituras y versiones originales de las visualizaciones. En esta sección evaluaremos el impacto de ambos enfoques en el esfuerzo dedicado a construir animaciones [236].

### 4.3.1. Diseño de la evaluación

Durante esta misma evaluación también investigamos aspectos pedagógicos, pero el problema que nos ocupa ahora es el esfuerzo de construcción, y no el aprendizaje de los alumnos. Para obtener datos sobre el esfuerzo necesario para la construcción de animaciones pedimos a los alumnos que realizaran un conjunto de tareas y respondieran a un cuestionario.

**Infraestructura.** La infraestructura es similar a la utilizada en la evaluación de R-Zoom (apartado 4.2.3). Los ordenadores eran los mismos, aunque en este caso el software de monitorización no era necesario. Al igual que en la evaluación de la técnica R-Zoom, los ordenadores eran suficientemente potentes como para ejecutar sin problemas el entorno de programación junto con sus capacidades de visualización.

**Participantes.** Los participantes en la evaluación eran 52 alumnos de la asignatura *Bases de lenguajes de programación*, donde se impartía programación funcional utilizando el entorno WinHIPE. Su participación era totalmente voluntaria y ninguno de ellos tenía problemas de visión que influyera en su trabajo con el entorno.

**Protocolo.** Durante el curso se organizaron siete sesiones de laboratorio. Las dos últimas se dedicaron a una clase de dudas y al examen de prácticas; las cinco anteriores se usaron para la evaluación. Durante las cuatro primeras sesiones se familiarizó a los alumnos con el entorno y sus facilidades –programación y visualización–, aunque no se dedicó tiempo explícitamente a la construcción de animaciones. En la quinta sesión se propusieron tres tareas: construcción de una animación (20 minutos), solución de un problema de programación (25 minutos) y depuración de un programa erróneo (25 minutos). Para terminar, contestaron al cuestionario del que sacaremos los datos sobre el esfuerzo.

### 4.3.2. Resultados de la evaluación

El objetivo de esta evaluación es analizar el esfuerzo dedicado por los estudiantes a construir animaciones. Para ello usamos tres preguntas del cuestionario antes mencionado:

1. *¿Con qué frecuencia has utilizado (antes del día de hoy) las facilidades de visualización que proporciona WinHIPE (evaluación gráfica, miniaturas, animaciones)?* Con esta pregunta sabemos la experiencia previa del estudiante con las facilidades de visualización.

2. *¿Te han resultado fáciles de usar las animaciones?* Obviamente, la facilidad de uso es un indicativo del esfuerzo dedicado a construir las animaciones.

3. *¿Has utilizado animaciones para depurar el algoritmo?* La tarea de depuración no requiere en sí el uso de las animaciones. Si se usan, significará que a los estudiantes no les supone mucho esfuerzo.

En cuanto al uso previo de las facilidades de visualización en WinHIPE, el 46,2 % no las ha usado nunca, el resto las ha usado en mayor o menor medida –esporádicamente, frecuentemente, o siempre que usa WinHIPE–. Las animaciones son claramente fáciles de usar; el 61,29 % está parcial o totalmente de acuerdo con esto, mientras que sólo el 6,45 % esta parcialmente en desacuerdo. Finalmente, el 51,9 % de los participantes usó las visualizaciones para depurar.

Si además añadimos que tanto la facilidad de uso como el uso para depuración, son independientes de la experiencia previa de las facilidades de visualización del entorno, podemos concluir que la construcción de animaciones con WinHIPE no requiere apenas esfuerzo, permitiendo a los estudiantes centrar su atención en las actividades de aprendizaje.

## 4.4. Resumen de aportaciones

En este capítulo hemos descrito un conjunto de aportaciones que ayudan a conseguir que el proceso de construcción de animaciones sea más sencillo. Para ello hemos atacado el problema desde dos puntos distintos: la generación de las miniaturas, y la selección de las visualizaciones para formar parte de la animación.

El proceso básico de generación de las miniaturas consiste en la reducción, en tamaño, de las visualizaciones. Nos hemos marcado como objetivo principal, mantener el máximo de propiedades de las visualizaciones en sus correspondientes miniaturas. Este objetivo lo hemos dividido en tres partes: relación de aspecto, redex de las visualizaciones y producción de las miniaturas. Al mantener la relación de aspecto, comunicamos de forma más fiel el contenido de las visualizaciones. Por otro lado, realzamos el redex de las visualizaciones, consiguiendo que sea más visible en las miniaturas, y por lo tanto manteniendo propiedades que informan sobre el flujo de ejecución. Finalmente, hemos diseñado y evaluado un algoritmo que realiza las reducciones de las visualizaciones en poco tiempo y obteniendo resultados de calidad significativamente mejores al estado anterior de WinHIPE[15].

Una vez que hemos conseguido miniaturas más comprensibles, hemos trabajado sobre el proceso de selección de las visualizaciones. El objetivo era mantener una visión global de la colección, generando las miniaturas, a la par que una detallada de las visualizaciones correspondientes. Para ello hemos diseñado una técnica de visualización de información llamada *R-Zoom*, basada en los principios de las técnicas *focus+context*. Hemos evaluado R-Zoom con éxito, donde el principal resultado es que los usuarios familiarizados con el tipo de tareas de nuestro dominio que trabajaron con R-Zoom, fueron más rápidos y cometieron menos errores. Además R-Zoom obtuvo la mejor valoración en cuanto a la satisfacción de los usuarios se refiere.

El resultado final es una interfaz que permite integrar una vista detallada de un momento de la ejecución del programa, junto con el resto del contexto. En la figura 4.36, podemos ver cómo se percibe sin problemas el flujo de ejecución, gracias a las miniaturas de alta calidad, a la vez que profundizamos en un paso en concreto para estudiarlo más en detalle. Como hemos comprobado, todo esto contribuye a que el proceso de construcción de las animaciones requiera poco esfuerzo por parte de los estudiantes, consiguiendo así que se centren en las actividades de aprendizaje.

[15]Hemos de reconocer que la idea de aplicar el algoritmo de Bresenham al escalado de imágenes fue propuesta, pero no evaluada, por Kientzle [111]. Nosotros descubrimos su trabajo después de diseñar el algoritmo.

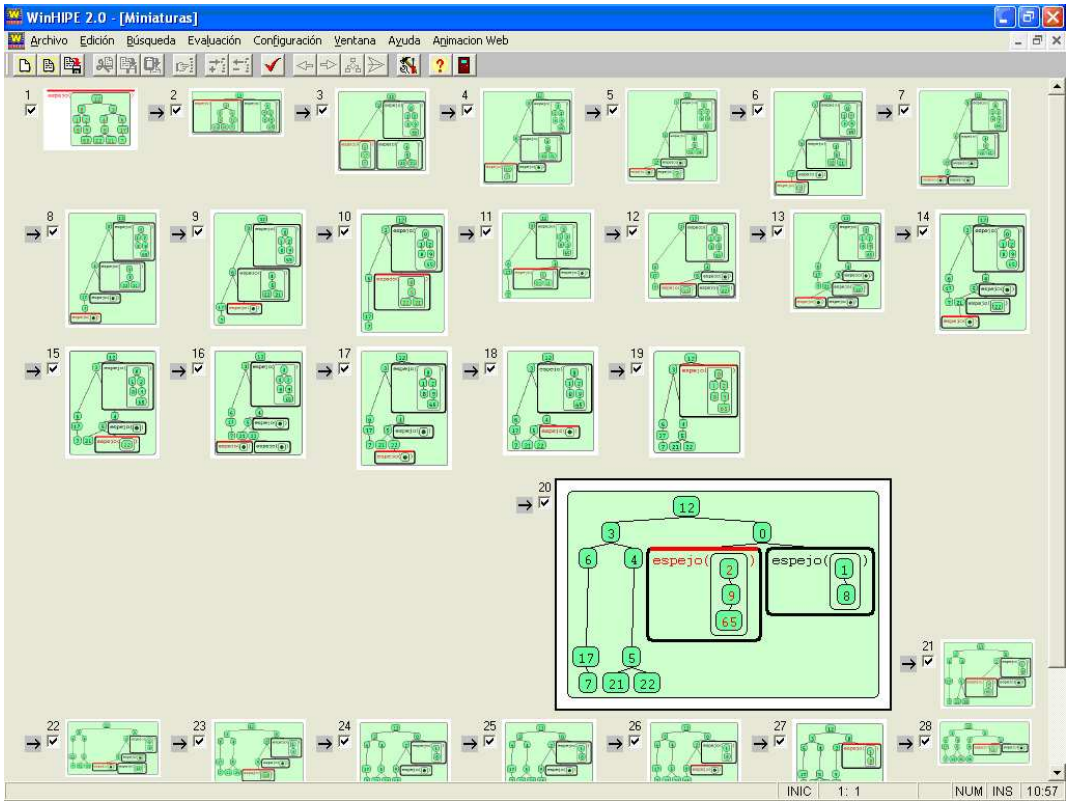



Figura 4.36: Integración de detalle (visualización) y contexto (miniaturas)

# Capítulo 5

# Uso educativo de las animaciones

Dentro de nuestro modelo de construcción de animaciones, una de las tareas fundamentales es la selección de los pasos de ejecución a mostrar en la animación; en el capítulo anterior hemos tratado las características técnicas que ayudan a conseguir una interfaz que facilite esta tarea. Sin embargo, se sigue generando el mismo tipo de animaciones, que entusiasma a los estudiantes al usarlas, pero cuya eficacia pedagógica no se ha podido demostrar [135]. Por ello, decidimos cambiar el diseño de las animaciones así como sus contenidos. El cambio más visible es que ahora las animaciones serán documentos publicables en la Web (en adelante nos referiremos a ellas como *animaciones Web*). Estos cambios nos han obligado a añadir ciertas características al proceso de construcción, y por lo tanto al entorno de programación, de forma que se mantenga la condición de facilidad en la construcción de animaciones.

El resto del capítulo se estructura como sigue. En primer lugar describiremos las animaciones Web, centrándonos en los contenidos, el proceso de construcción, y su diseño gráfico. Después trataremos el modelo para la construcción y mantenimiento de colecciones de estas animaciones Web. A continuación, describiremos los posibles usos educativos de las animaciones Web, detallando las evaluaciones que hemos realizado del uso de las animaciones Web por parte de los estudiantes. Finalmente, resumimos las aportaciones descritas en este capítulo.

## 5.1. Animaciones Web

La publicación de animaciones de algoritmos en la Web es un tema ya tratado por otros autores [170]. Algunos de los sistemas de animación basados en la Web son applets diseñados ad-hoc para visualizar determinados algoritmos [59, 79, 152, 218]. Otros sistemas, más flexibles, permiten al usuario la generación de sus propias animaciones [2, 65, 115, 147, 156, 192, 191, 220]. Las animaciones Web tratadas en este capítulo pertenecen a este último grupo.

La idea de la generación automática de animaciones Web con WinHIPE se describió por primera vez en [147]. El resultado era una página Web dinámica, cuya estructura era una lección sobre un determinado algoritmo. Dicha estructura estaba compuesta de cuatro secciones (véase un ejemplo en la figura 5.1). Las dos primeras secciones describían el problema y el algoritmo que lo resolvía; estas secciones se generaban con un simple comentario, siendo el usuario el responsable de su edición en la página Web. Las secciones tercera y cuarta contenían respectivamente el código fuente del programa que posteriormente sería animado, y la animación propiamente dicha.

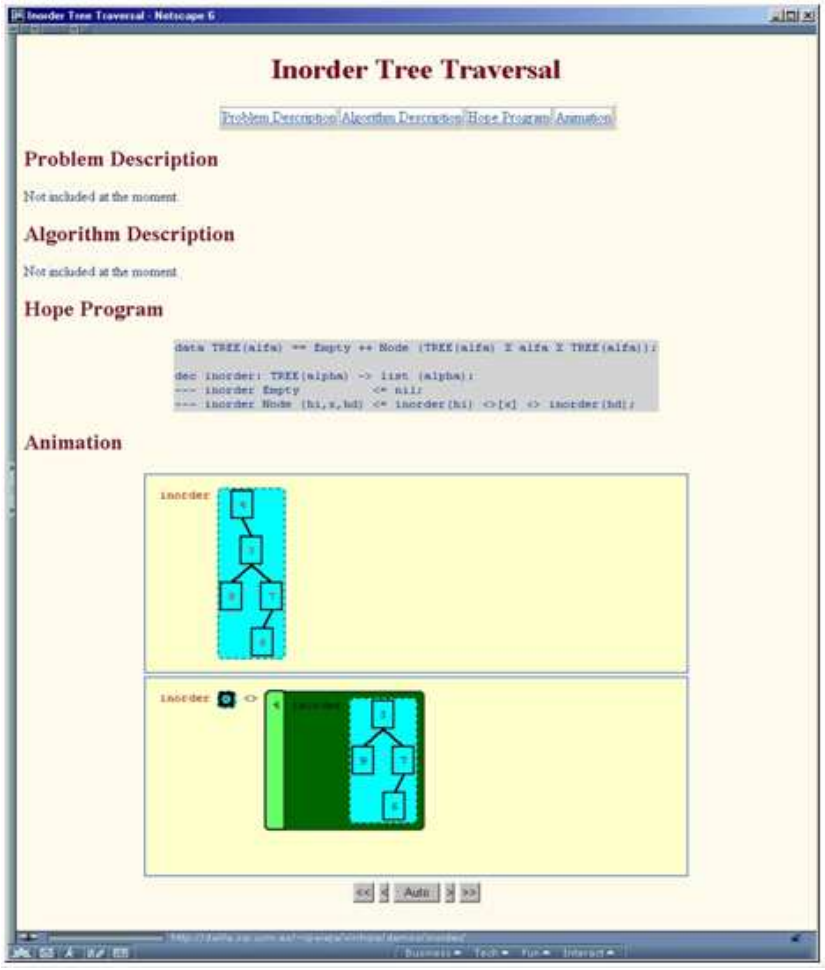



Figura 5.1: Un ejemplo de las primeras animaciones Web generadas con WinHIPE. Adaptado de [147], cortesía de J. Ángel Velázquez Iturbide

Tanto el texto como el diseño gráfico de la página Web se podían cambiar con un editor de HTML, pero estaríamos ignorando una de las características que ayudan a la eficacia pedagógica, la integración de las animaciones en el entorno de programación. Además, si el usuario quería cambiar la animación, debía construirla entera desde el principio, y aunque esta tarea no es muy costosa, la aplicación no proporcionaba ninguna ayuda para realizarla. Para permitir realizar cambios sobre las animaciones, bastaría con facilitar tareas simples, como la búsqueda de los ficheros de programa y expresión, o la utilización de la información de personalización de la animación –selección de fotogramas y configuración gráfica de las visualizaciones estáticas–, tareas cuya mecanización es sencilla.

Estamos especialmente interesados en ofrecer una herramienta que permita construir animaciones Web con poco esfuerzo, pero que además sean herramientas pedagógicas eficaces. En consecuencia, hemos rediseñado y reimplementado la parte de WinHIPE dedicada a la construcción de animaciones Web. Hemos cambiado los contenidos de las páginas Web y su proceso de construcción, y también hemos enriquecido su diseño gráfico mejorando su usabilidad.

### 5.1.1. Composición de las animaciones Web

Hemos cambiado los contenidos de las animaciones Web para que su proceso de modificación sea más fácil de cara al usuario. Las animaciones Web antiguas eran una página Web dinámica escrita en HTML y JavaScript. Ahora, una animación Web contiene toda la información necesaria para realizar cambios sobre ella. Estos cambios pueden influir en su apariencia, secciones de contenido textual, e incluso la propia animación.

Las animaciones Web se componen de información de dos clases diferentes. En primer lugar tenemos la información visible, que es aquella que será procesada por el navegador Web para visualizar la animación Web. Esta información estará compuesta de código XHTML, estilos CSS e imágenes. En segundo lugar tenemos la información de mantenimiento, que es aquella información necesaria para la gestión y modificación de las animaciones Web. Está información se almacenará en XML; su DTD se puede ver en el apéndice B.1, y un ejemplo de dicha información se encuentra en el anexo B.2.

Los contenidos de una animación Web se dividen en tres partes: texto, animación y apariencia. Los contenidos de texto tienen la información referente a las descripciones del problema y del algoritmo. Los contenidos de animación tienen información referente a la construcción y reproducción de la animación: código fuente, expresión evaluada, lista de fotogramas que forman la animación, y los datos de configuración que describen la representación gráfica de las visualizaciones. Finalmente, los contenidos de apariencia describen el aspecto de la página Web que contiene la animación Web. En la figura 5.2 se puede ver un esquema de los contenidos y la información de una animación Web.

| | | ***Contenido*** | | |
|---|---|---|---|---|
| | | ***textual*** | ***animación*** | ***apariencia*** |
| ***Información*** | ***mantenimiento*** | Título<br>Problema<br>Solución | Código fuente<br>Expresión<br>Fotogramas<br>Estilo de las visualizaciones | Estilo de la página web |
| | ***visual*** | <title><br><p> | <code><br><img> | css |

Figura 5.2: Información y contenidos que componen una animación Web

Nótese que las animaciones Web son reutilizables desde dos puntos de vista diferentes. Por un lado, se pueden reutilizar desde el punto de vista del usuario, ya que contienen toda la información necesaria para modificarlas, haciéndolas totalmente independientes de quién las creó. Por otro lado, son reutilizables desde el punto de vista de la plataforma, tanto a nivel de mantenimiento, ya que su implementación se hace con XML/XHTML, y por lo tanto, no sólo se pueden usar en el entorno WinHIPE, sino en cualquier otra aplicación que pueda manipular documentos XML; como a nivel de uso, ya que son documentos manipulables por cualquier navegador Web.

En resumen, la composición de las nuevas animaciones Web reduce el esfuerzo que un usuario debe dedicar a su modificación y mantenimiento, ya que no tiene que recordar ninguna información sobre contenidos o configuración. Y además, son independientes de la plataforma.

### 5.1.2. El proceso de construcción

El proceso de construcción de las animaciones Web se divide en dos fases: generación de la animación del programa y generación de la animación Web. La figura 5.3 ilustra el proceso completo.

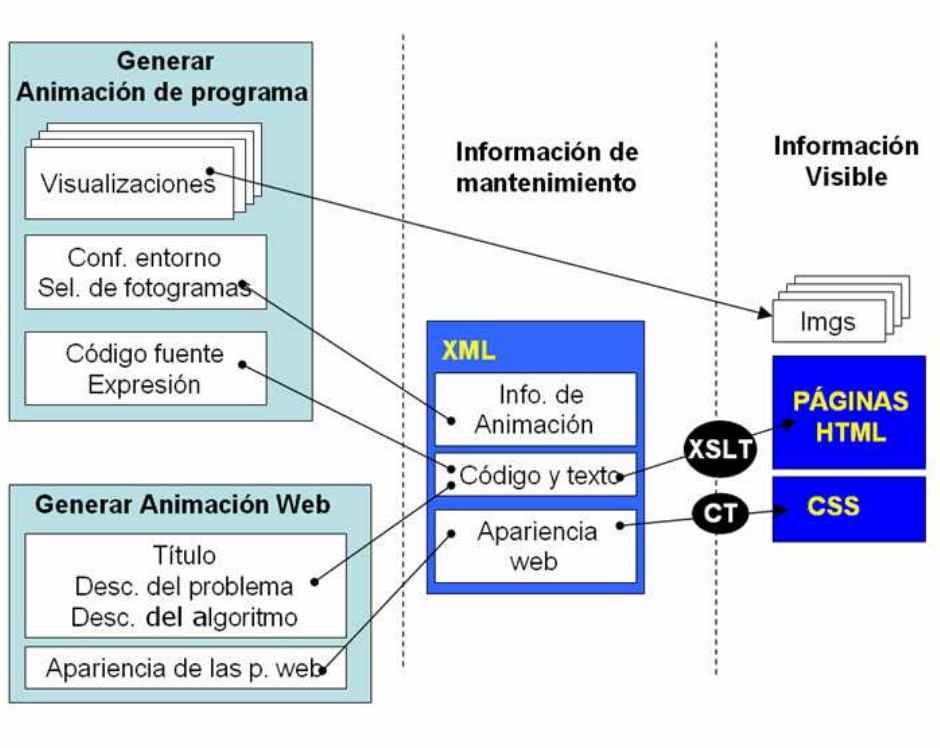



Figura 5.3: El proceso de construcción de las animaciones Web

En términos generales, la fase de *generación de la animación del programa* se compone de los siguientes pasos: edición y compilación del programa, edición y evaluación de la expresión, y selección de los fotogramas que compondrán la animación. Las dos primeras son tareas rutinarias que haría cualquier programador. Si a esto añadimos la interfaz presentada en el capítulo anterior para la selección de los fotogramas de la animación, podemos concluir que la fase de generación de la animación del programa es sencilla y requiere del usuario poco esfuerzo para completarla.

La *generación de la animación Web* se hace con ayuda del propio entorno WinHIPE. El usuario debe introducir la información de mantenimiento relativa a los contenidos de texto y apariencia; para ello hemos diseñado una interfaz sencilla que ayude en esta tarea (véanse las figuras 5.4 y 5.5). La información de texto se compone del título de la animación Web, la descripción del problema y la del algoritmo. La información de apariencia es el estilo con el que se generará la animación Web, que consistirá en las propiedades tipográficas de la información visual, así como las diferentes formas de ver la propia animación.


Figura 5.4: Interfaz de generación de animaciones Web, encabezado principal de la animación Web. La interfaz permite introducir la información de mantenimiento sobre contenidos textuales (zona 1), y contenidos de apariencia (zona 2). Además proporciona una vista previa del resultado, la información visual (zona 3)


Figura 5.5: Interfaz de generación de animaciones Web, secciones que componen la animación. En concreto aquí mostramos la parte relacionada con la sección de la descripción del problema. Se puede manipular cualquier sección (zona 1). En este caso, la interfaz permite introducir la información de mantenimiento sobre contenidos textuales (zonas 2 y 4), y contenidos de apariencia (zona 3). Además, proporciona una vista previa del resultado, la información visual (zona 5)

Todos los datos generados durante la construcción de la animación del programa, junto con la información proporcionada por el usuario durante la generación de la animación Web, forman parte de un documento XML donde se encuentra la información de mantenimiento junto con la identificación de los fotogramas que forman la animación.

Es importante destacar que, hasta ahora, el usuario ha seguido los mismos pasos que daría para construir cualquier animación. Además, deberá añadir las explicaciones del problema y la solución, y deberá especificar la apariencia de la animación Web. A partir de este momento el usuario ha terminado su trabajo, el resto del proceso es automático.

Para terminar, el proceso automático genera la información visible, excepto los fotogramas, que se han generado previamente. La información visible de los contenidos textuales y la animación, se generan –utilizando trasformaciones XSL (XSLT)– en forma de código XHTML (distribuido en una o varias páginas Web, como se verá en el siguiente apartado). La información visible correspondiente a la apariencia será construida mediante una transformación ad-hoc (CT) cuyo resultado es una hoja de estilos CSS.

### 5.1.3. Diseño gráfico de las páginas Web

En la primera versión de las animaciones Web [147], se utilizaba una página Web donde se podía navegar a través de enlaces locales y movimientos verticales de la barra de desplazamiento. Dicha página contenía la información visible correspondiente a los contendidos de texto y animación (véase figura 5.1).

Hemos enriquecido este formato para permitir diferentes presentaciones de las animaciones Web. Dos de estos formatos se inspiran en principios generales de usabilidad de páginas Web [162]. Llamamos a estos formatos *Framed1* y *Framed2*. El tercero se diseñó pensando en ofrecer la máxima flexibilidad al usuario a la hora de ver la animación Web; le llamamos *Star*.

El usuario puede acceder en todo momento a los cuatro formatos, los tres anteriores junto al antiguo, mediante una barra de navegación (véase figura 5.6) situada en la parte superior izquierda de la animación Web. Además, con esta barra de navegación también se permite el acceso a los contenidos de la animación Web codificados en XML.


Figura 5.6: Barra de navegación. Compuesta por cinco iconos que permiten acceder a (de izquierda a derecha) los formatos *antiguo*, *Framed1*, *Framed2*, *Star* y la información de mantenimiento

Los principales requisitos de usabilidad que se han considerado son: minimizar la cantidad de información oculta en la página (barra de desplazamiento vertical), ajustar el espacio ocupado por las facilidades de navegación a un máximo del 20 % del espacio disponible en la pantalla, tiempos de respuesta predecibles (sabiendo que siempre dependen del número y tamaño de los fotogramas de la animación), utilización de hojas de estilo CSS enlazadas, y utilización de marcos embebidos.

El formato *Framed1* (véase figura 5.7) usa un marco embebido, permitiendo al usuario acceder a cada sección de la animación Web con un solo clic de ratón. La barra de desplazamiento aparece sólo cuando el contenido de la sección es mayor que el espacio disponible dentro del marco embebido. Tanto

el título, como los enlaces directos a las secciones y la barra de navegación están permanentemente visibles en la pantalla.

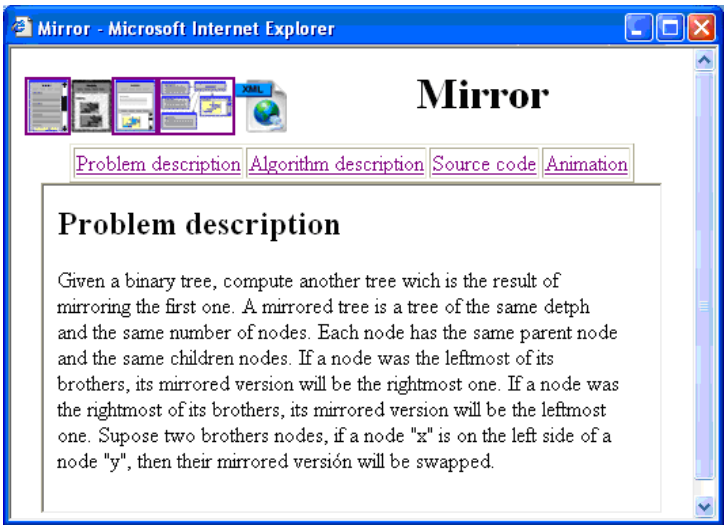



Figura 5.7: Animación web con formato Framed1

El formato *Framed2* (véase figura 5.8) es similar a *Framed1*, pero con dos marcos embebidos, en vez de uno solo. Permite acceder de forma simultánea a dos secciones de la animación Web.

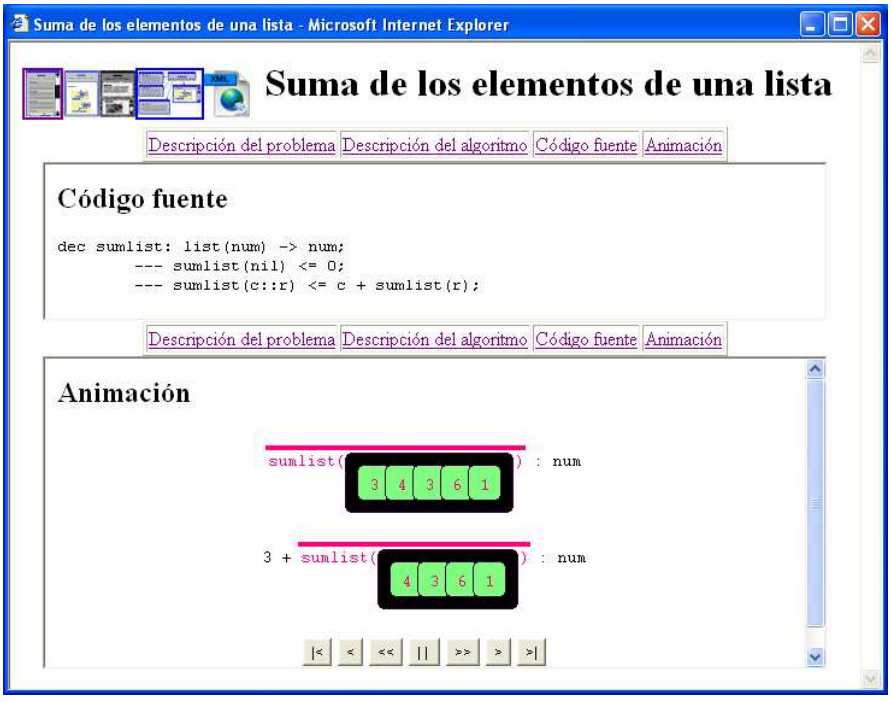



Figura 5.8: Animación web con formato Framed2

Finalmente, el formato *Star* (véase figura 5.9) permite al usuario acceder de forma simultánea a tantas secciones como desee. Consiste en una pequeña página Web con el título, la barra de navegación y los enlaces a las secciones de la animación. Cada sección se muestra en una página Web diferente, siendo el usuario quien adapta las dimensiones de cada página a sus necesidades.

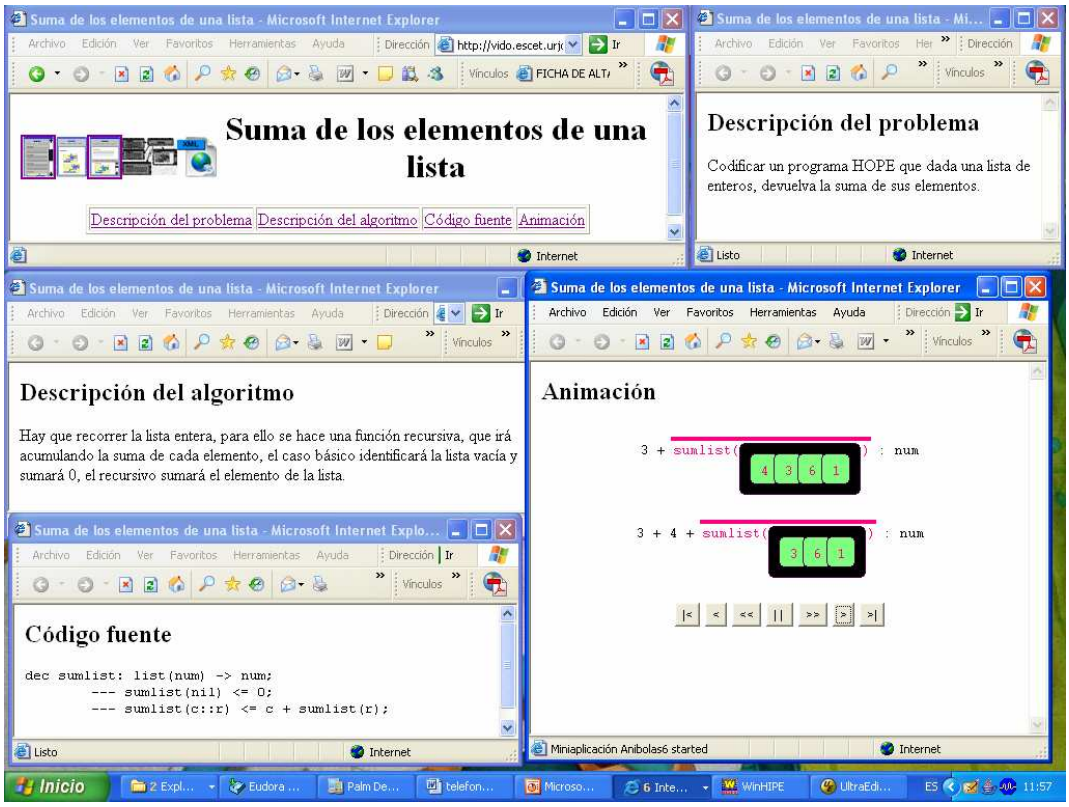



Figura 5.9: Animación web con formato Star

### 5.1.4. Estructura final de una animación Web

Una vez conocidos los contenidos, el proceso de construcción y el diseño gráfico de las animaciones Web, podemos identificar cuál es su estructura final. Una animación Web consiste en toda la información de mantenimiento codificada en un documento XML, los fotogramas que forman la animación, y las páginas Web correspondientes a los cuatro formatos de presentación: *antiguo*, *framed1*, *framed2* y *star*. Todos estos componentes se guardan en un mismo directorio (véase figura 5.10).

## 5.2. Un modelo para las colecciones de animaciones

Hasta el momento, hemos hablado de las animaciones como elementos individuales. Sin embargo, un profesor no se parará en desarrollar una o dos animaciones; probablemente termine construyendo colecciones de animaciones que cubran parte de los conceptos que quiere explicar. La inclusión de las animaciones en la metodología pedagógica implica ciertas tareas como: encontrar el sistema de animación, instalarlo, aprender a usarlo, construir las animaciones, y otras tareas rutinarias menores. Es normal que los profesores quieran rentabilizar todo el tiempo invertido en estas tareas, usando las animaciones para más de un tema concreto [158]. Así pues, el profesor podría terminar desarrollando y

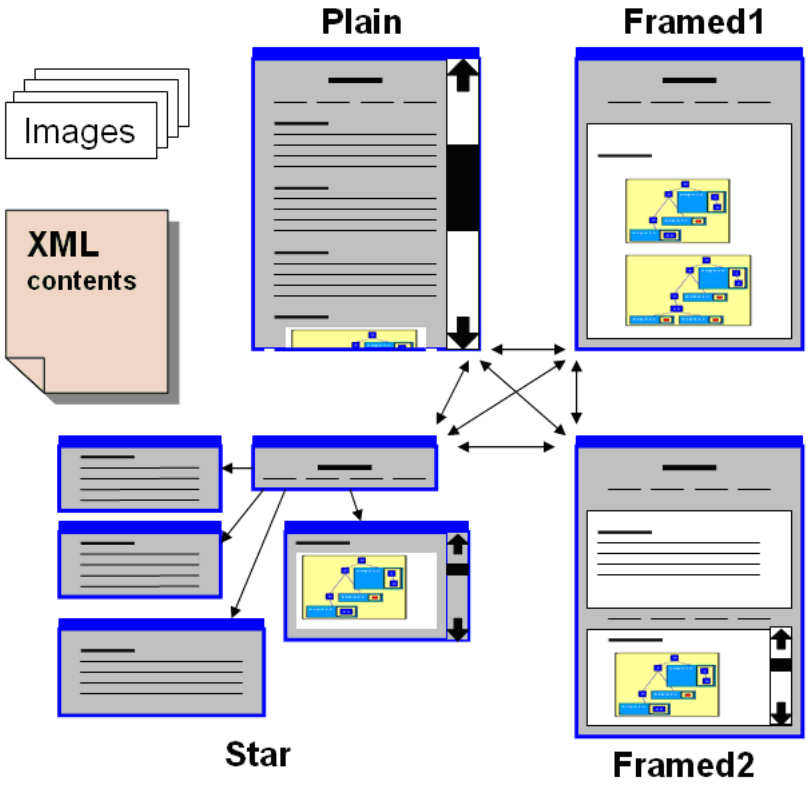



Figura 5.10: Estructura final de una animación Web, con representaciones esquemáticas de los cuatro formatos posibles para la animación Web

manteniendo, tanto a nivel individual como organizativo, colecciones de animaciones. Además, existen tareas típicas como la modificación de una o varias animaciones en cuanto a contenidos, estilo de presentación, o su lugar en la colección. En esta sección describimos un modelo, y su implementación parcial en un prototipo para la construcción y mantenimiento de colecciones de animaciones Web desarrolladas con nuestro entorno.

Definimos una colección como un conjunto animaciones Web sobre un tema común: una asignatura, una clase de problemas, etc. El usuario podrá crear tantas colecciones independientes como crea necesarias; cada una de ellas se organizará de forma jerárquica. Estas jerarquías no tendrán límite en cuanto a profundidad, ni número de categorías diferentes. Tanto la estructura jerárquica de las colecciones, como la naturaleza de sus contenidos, hacen que la tecnología XML sea la ideal para su implementación.

### 5.2.1. Gestión y construcción de colecciones de animaciones Web

La creación de una colección tiene dos escenarios posibles: partir de una colección vacía, que al menos deberá tener una categoría con su nombre asociado; o partir de un subconjunto de categorías de otra colección existente que, según el usuario, han evolucionado lo suficiente como para tener entidad propia como colección[1].

La gestión de las colecciones consiste en la gestión de su estructura jerárquica, las propias anima-

[1]Nótese que el criterio queda del lado del usuario, podría ser por razones de espacio, usabilidad, o reutilización de contenidos.

ciones Web, y su apariencia. La gestión de la estructura jerárquica comprende la inserción, eliminación o modificación de categorías. Cuando insertamos una categoría debemos proporcionar un nombre, una breve descripción y el punto de inserción en la jerarquía. En el caso de eliminación de categorías no vacías, debemos preguntar al usuario qué quiere hacer con sus contenidos: borrarlos, o trasladarlos a otro lugar. Finalmente, la modificación de una categoría puede traer cambios en cuanto a la descripción, el nombre, o el punto de inserción en la jerarquía; en estos dos últimos casos comprobamos la existencia de conflictos con sus categorías hermanas. Además, la gestión de una colección también permite la inserción de nuevas animaciones, su modificación o su eliminación, ya sea por traslado a otra categoría o por eliminación total de la colección.

Para mejorar la homogeneidad de la colección, cada categoría podrá tener asociada una apariencia. Así, esta apariencia se podrá aplicar a todas las animaciones de dicha categoría, e incluso a todas las categorías descendientes de ella. Pero también se debe permitir asociar apariencias particulares a determinadas animaciones. Obviamente, los cambios realizados sobre la estructura de la colección afectan a la apariencia, así que se pedirá al usuario que elija entre la apariencia original o la de la categoría de destino, si es que existe.

Todas estas operaciones se llevan a cabo mediante una interfaz simple (véase la figura 5.11), muy similar a las interfaces de navegación por sistemas de ficheros. El panel izquierdo muestra la estructura jerárquica de la colección. En la parte derecha tenemos dos paneles; el superior muestra los contenidos de la categoría seleccionada en la parte izquierda –categorías hijas y animaciones–, y el panel inferior permite la visualización y modificación de la animación Web seleccionada en el panel superior.

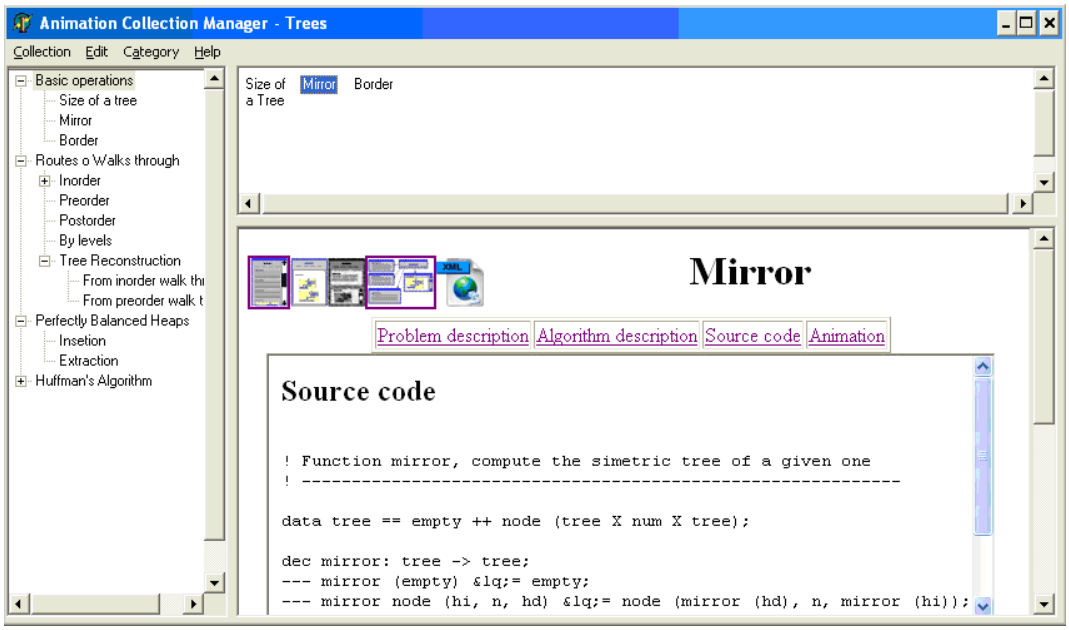



Figura 5.11: Interfaz del gestor de colecciones

### 5.2.2. Publicación Web de las colecciones de animaciones Web

La publicación en la Web de estas colecciones consiste en la transformación de su estructura jerárquica interna –implementada en XML– en un sitio Web, mediante trasformaciones XSL. Creamos una página raíz con la descripción de la colección y enlaces a las categorías del primer nivel. A con-

tinuación, por cada categoría creamos una página Web con el nombre de la categoría como título, su descripción, y los enlaces asociados a las animaciones y categorías hijas. Por otro lado, se puede crear un mapa que permita acceder directamente a una categoría de la colección. Y finalmente, para facilitar la navegación, enlazamos cada página (categoría), con su categoría padre, la página raíz, así como el mapa de la colección.

## 5.3. Uso educativo de las animaciones Web

En primera instancia, podríamos pensar en un escenario de uso típico. El profesor se encarga de las tareas de generación de animaciones y las usa como apoyo para sus explicaciones, y el alumno las utiliza como material de consulta. Basándose en este escenario, podemos encontrar trabajos como los citados en el estado de la cuestión, que tratan obtener el mayor beneficio posible de las animaciones, tanto disminuyendo el esfuerzo dedicado por los profesores a la hora de generarlas, como enriqueciendo las animaciones con explicaciones y preguntas.

En el escenario anterior, hemos asignado al profesor el rol de emisor, y a los estudiantes el rol de meros receptores. Sin embargo, existen otros puntos de vista que cambian estos roles, sobre todo el de los estudiantes. Desde el punto de vista de la teoría del constructivismo, una de las labores del profesor es proporcionar al estudiante experiencias que le permitan construir sus propios conocimientos. Así, se obliga al estudiante a involucrarse más con los materiales educativos. En nuestro caso, trataríamos involucrar a los estudiantes más allá de la simple consulta de las animaciones. Este cambio de rol se puede observar en los tres últimos niveles de implicación expuestos por Naps et al.[158]: *cambio*, donde el estudiante pasa a explorar el comportamiento del programa, proporcionando datos de entrada y observando los cambios producidos en las distintas ejecuciones del programa; *construcción*, donde el estudiante debe generar una representación del funcionamiento del programa (nótese que el estudiante empieza a tomar el papel del profesor, puesto que la animación debe expresar el comportamiento del programa de forma correcta y completa); y *presentación*, donde el estudiante presenta a una audiencia la animación, tomando totalmente el rol del profesor.

El trabajo de esta tesis está enfocado a la generación sin esfuerzo de animaciones de programas funcionales. Sin embargo, en vez de medir el esfuerzo dedicado por los profesores a la generación de animaciones, mediremos la eficacia pedagógica de este proceso de construcción de animaciones desde el punto de vista de los estudiantes. Al ser un enfoque sin esfuerzo, los estudiantes centrarán su atención en los contenidos que se presentarán en la animación, y no en el proceso de construcción de la misma. Desde el punto de vista constructivista, la construcción sin esfuerzo podría ser una experiencia que permitiera a los estudiantes construir su conocimiento sobre los aspectos de programación funcional tratados en cada animación. A continuación presentamos la evaluación de este enfoque de construcción sin esfuerzo. En primer lugar hacemos una evaluación a corto plazo donde medimos la eficacia pedagógica en el aprendizaje de un algoritmo concreto. En segundo lugar, y aprovechando la experiencia de la primera evaluación, hacemos una evaluación a largo plazo donde comparamos tres escenarios distintos: no utilizar animaciones, verlas y construirlas; analizando su impacto en el aprendizaje y la actitud de los estudiantes en una asignatura sobre programación funcional.

## 5.4. Evaluación a corto plazo

En esta evaluación investigamos los efectos de la construcción de animaciones de algoritmos en su aprendizaje. Usando la taxonomía de niveles de implicación con las animaciones de Naps et al [158], hemos comparado el nivel de construcción –implementado con nuestro enfoque de *construcción sin esfuerzo*–, con el nivel de visión. Mediremos dichos efectos en términos de la taxonomía de Bloom [28], en concreto, en los niveles de comprensión y aplicación.

El contexto de esta evaluación es la asignatura de *Diseño y Análisis de Algoritmos* de la titulación de *Ingeniería Informática* de la *Universidad Rey Juan Carlos*. Dividimos a los alumnos de la asignatura en dos grupos: uno que vio un conjunto de animaciones, y otro que construyó animaciones utilizando nuestro enfoque *sin esfuerzo*. La evaluación se centró en el algoritmo de recorrido de árboles en anchura.

### 5.4.1. Participantes

Los participantes eran 15 estudiantes voluntarios (13 varones y 2 mujeres) de la asignatura antes mencionada. Los dividimos de forma aleatoria en dos grupos: los *visores* (GV), que sólo vieron las animaciones; y los *constructores* (GC), que construyeron las animaciones. Todos los participantes respondieron a un test de conocimientos sobre el algoritmo, y sólo un alumno demostró tener algún conocimiento, por lo que ambos grupos pertenecen a la misma población y la comparación de sus resultados será válida.

### 5.4.2. Variables

La variable independiente de la evaluación es el nivel de implicación con las animaciones, mientras que las variables dependientes son la eficacia y eficiencia pedagógicas, así como la opinión de los estudiantes.

Para medir la eficacia pedagógica usamos un test de conocimientos sobre el algoritmo (véase apéndice C.1). Diseñamos las preguntas de este test según la taxonomía de Bloom [28], así las cuatro primeras preguntas se asocian al nivel de comprensión, mientras que la quinta pregunta se asocia al nivel de aplicación. La eficiencia pedagógica la medimos como el tiempo empleado en estudiar el algoritmo y responder al test de conocimientos. Finalmente, la opinión de los estudiantes la medimos con dos cuestionarios, uno para los visores y otro para los constructores (véase apéndice C.2).

### 5.4.3. Protocolo

Dividimos la evaluación en dos sesiones: la primera, donde se enseña a los estudiantes el entorno; y la segunda, donde evaluamos el conocimiento sobre el algoritmo y la opinión de los estudiantes. En la primera sesión participaron diez estudiantes, en la segunda trece.

El objetivo de la primera sesión, de dos horas de duración, era familiarizar a los estudiantes con el entorno. Durante esta sesión el profesor hizo una demostración del entorno, primero generó dos animaciones a modo de ejemplo, y a continuación, los estudiantes generaron dos animaciones más. Las animaciones con las que se trabajó no estaban relacionadas con el algoritmo usado en la segunda sesión. Ningún estudiante tuvo problemas con el entorno. Al final de esta sesión, pedimos a los estudiantes que respondieran a un cuestionario de opinión sobre el entorno, así obtuvimos su primera impresión sobre la herramienta.

En la segunda sesión realizamos las mediciones sobre la eficacia y eficiencia pedagógicas, así como la opinión de todos los estudiantes. Esta sesión también duró dos horas, y tuvo lugar dos semanas después de la primera. En primer lugar les explicamos a los estudiantes que íbamos a realizar la evaluación y que su participación sería voluntaria. A continuación dividimos a los participantes en los dos grupos: GV ($n = 7$) y GC ($n = 6$). Además pudimos comprobar que todos los estudiantes del grupo GC habían participado en la primera sesión, así que sus resultados no se verían afectados por el desconocimiento del entorno. También comprobamos el conocimiento previo de ambos grupos sobre el algoritmo. Después les dimos a los estudiantes los materiales que debían utilizar para estudiar el algoritmo. Los materiales de estudio constaban de una explicación textual del mismo (véase apéndice C.3), junto con:

- un conjunto de animaciones Web construidas con WinHIPE, para el grupo GV, y
- el código fuente del algoritmo para construir animaciones del mismo, para el GC.

Les pedimos a los participantes que estudiaran el algoritmo con los materiales facilitados durante el tiempo que ellos creyeran necesario. Los participantes sabían que después contestarían al test de conocimientos. Finalmente, tuvieron que rellenar el test de conocimientos, y los cuestionarios donde les pedimos su opinión sobre la experiencia de aprendizaje viendo, o construyendo, animaciones Web.

### 5.4.4. Resultados de la evaluación

A continuación explicamos los resultados de la evaluación desde los tres puntos de vista antes mencionados, eficacia y eficiencia pedagógica, así como la opinión subjetiva de los estudiantes. Los detalles del estudio estadístico se encuentran en el apéndice C.4.

#### Eficacia pedagógica

Medimos la eficacia en el aprendizaje según dos niveles de la taxonomía de Bloom: comprensión y aplicación. La puntuación obtenida en ambos niveles pertenecía al rango $[0, 1]$. En la Fig. 5.12 mostramos un resumen de cada pregunta del test de conocimientos.

Las preguntas relacionadas con el nivel de comprensión eran cuatro. En la primera, pedimos a los estudiantes que identificaran las ideas principales del algoritmo, todas ellas visibles en las animaciones; estas ideas eran: (1) operaciones con listas, (2) el recorrido del árbol se hace de izquierda a derecha, y (3) para hacer el recorrido acumulamos las operaciones pendientes con cada subárbol. Los estudiantes de ambos grupos mencionaron por igual las ideas (1) y (2), mientras que la idea (3) sólo fue identificada por un 14 % (1 de 7) de los estudiantes en el GV –a pesar de estar explícitamente mencionado en la descripción textual del algoritmo– contra un 83 % (5 de 6) de los estudiantes en el GC (diferencia estadísticamente significativa $U = 7{,}000, p = \mathbf{0.013}$). Sin embargo, teniendo en cuenta las cuatro preguntas del nivel de comprensión, no hemos detectado diferencia significativa en el aprendizaje.

En la pregunta asociada al nivel de aplicación detectamos diferencias significativas $t(5{,}376) = 2{,}915, p = \mathbf{0.03}$. El GV obtuvo una puntuación de 0,33, contra un 0,77 del GC, un 60 % de mejora. Debemos destacar que la mejora a nivel de aplicación tiene, en parte, su origen en el nivel de comprensión; ya que todos aquellos estudiantes que no detectaron la idea principal (3) no sugirieron su cambio, obteniendo una puntuación menor en el nivel de aplicación.

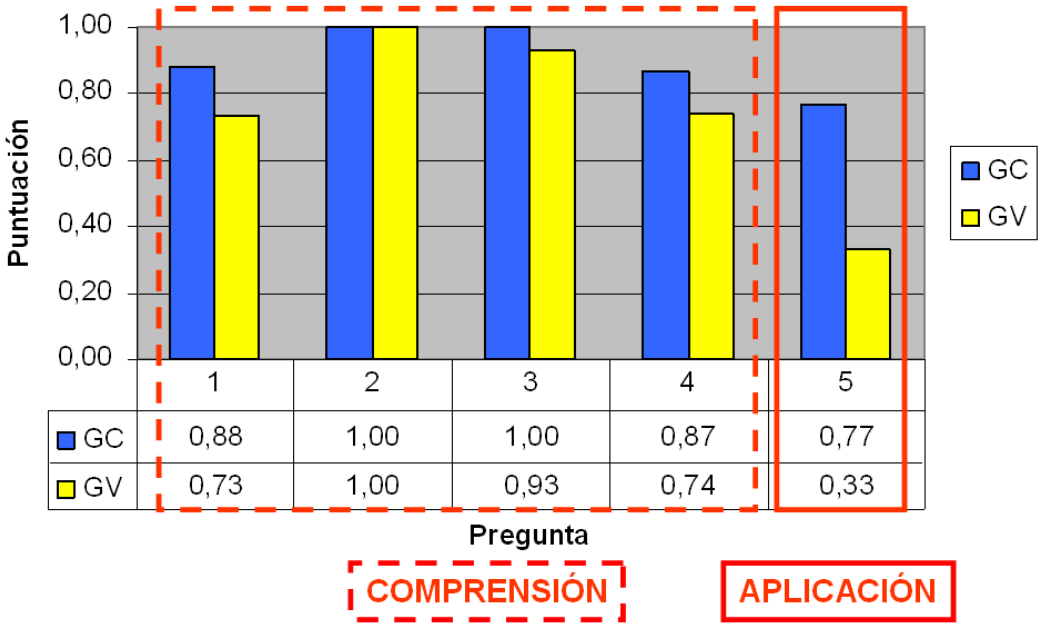



Figura 5.12: Puntuaciones obtenidas en el test de conocimientos. El recuadro rojo de línea discontinua identifica a las preguntas, de la 1 a la 4, relacionadas con el nivel de comprensión. El recuadro rojo de línea continua identifica a la pregunta relacionada con el nivel de aplicación

### Eficiencia pedagógica

Hemos medido la eficiencia pedagógica en función del tiempo empleado por los participantes en estudiar el algoritmo, utilizando los materiales proporcionados, así como el tiempo empleado en responder al test de conocimientos.

En cuanto al tiempo empleado por los estudiantes en estudiar el algoritmo, el GC dedicó una media de 49 minutos ($M = 49{,}0, SD = 4{,}97$), mientras que el GV dedicó una media de 18 minutos ($M = 18{,}0, SD = 5{,}41$). Evidentemente, el tiempo dedicado al estudio es significativamente diferente, $t(11) = 10{,}670, p = \mathbf{0.000}$.

En cuanto al tiempo empleado en responder al test de conocimientos no detectamos diferencias significativas, $t(11) = 1{,}384, p = 0{,}194$, entre los estudiantes del GV ($M = 15{,}4, SD = 5{,}79$), y del GC ($M = 19{,}6, SD = 5{,}12$).

### Satisfacción de los estudiantes con el entorno

La primera impresión de los estudiantes (cuestionario de opinión contestado tras la primera sesión), tanto con WinHIPE, como con el proceso de construcción, fue muy satisfactoria. Ninguno de ellos había usado WinHIPE anteriormente. Todos ($n = 10$) opinaron que las animaciones Web eran fáciles de construir, y que el hecho de construir animaciones les ayudaba a entender los algoritmos.

Los estudiantes *constructores* mantuvieron esta opinión después de la segunda sesión. Además, todos los *visores* estaban de acuerdo con que: las animaciones Web les ayudaba a comprender el algoritmo, eran útiles y fáciles de usar. Además, pedimos a los estudiantes constructores que opinaran sobre ambos enfoques, preguntándoles cuál de ellos creían más apropiado para aprender el algoritmo: visión o construcción. El 71 % (5/7) opinaban que ambos enfoques eran apropiados, tres de ellos además dijeron que sería bueno utilizar ambos enfoques juntos.

### 5.4.5. Interpretación de los resultados

Creemos que existen dos factores origen de las mejoras detectadas en el aprendizaje de los constructores: el nivel de implicación, y el tiempo empleado en el estudio. Con los datos obtenidos en esta evaluación, no tenemos forma de diferenciar cuál de los dos factores es más influyente. Una primera idea podría ser que nuestro enfoque de construcción de animaciones consigue implicar a los alumnos constructores mucho más, por lo tanto los constructores dedican más tiempo a estudiar el algoritmo, lo que probablemente sea un factor clave en la obtención de mejoras en el aprendizaje.

Pero según esta interpretación, estamos suponiendo que construir animaciones es una tarea de la misma naturaleza que verlas, y no es así. El proceso de construcción de animaciones conlleva que el estudiante decida si una determinada visualización debe aparecer en la animación o no, lo cual requiere de un conocimiento más profundo del algoritmo. Las tareas de visión de animaciones tan sólo requieren del alumno controlar la velocidad y dirección de la animación, dejando al alumno la libertad de estudiar más en detalle los pasos de ejecución mostrados. Obviamente, la construcción lleva más tiempo al estudiante, pero también le obliga a profundizar en el estudio del algoritmo, lo que también mejora el aprendizaje sobre el mismo.

Creemos que el nivel de implicación ha afectado a las mejoras detectadas en el aprendizaje, tanto directa, como indirectamente (véase Fig. 5.13). Directamente, ya que la construcción requiere del estudiante un conocimiento más profundo del algoritmo. E indirectamente, ya que la construcción lleva más tiempo de trabajo, y por lo tanto más tiempo de estudio del algoritmo, facilitando un mejor aprendizaje del mismo.

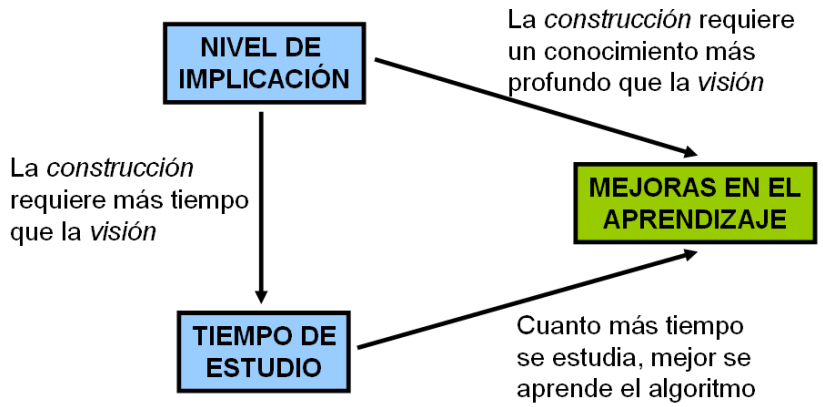



Figura 5.13: Influencia del nivel de implicación y el tiempo de estudio, sobre las mejoras en el aprendizaje del algoritmo

Por otro lado, creemos que la generalización de este resultado es muy limitada; ya que se restringe a un algoritmo concreto, un periodo de tiempo corto, y una población pequeña. Aún así, creemos que la construcción de animaciones ha influido positivamente en el aprendizaje de los estudiantes.

La siguiente evaluación trata de evitar los tres inconvenientes antes detectados. Así, ampliaremos el ámbito de aplicación –en tiempo y materia– a todo un curso básico de programación funcional, y aumentaremos el número de estudiantes, incluyendo un grupo de control que no use animaciones de ninguna forma.

## 5.5. Evaluación a largo plazo

En esta evaluación investigamos los efectos de la visión y la construcción de animaciones de programas funcionales como herramientas educativas. Usando la taxonomía de niveles de implicación con las animaciones de Naps et al. [158], hemos comparado el nivel de *construcción* –de nuevo implementado con nuestro enfoque de construcción sin esfuerzo–, con los niveles *sin visión* y *visión*. Medimos dichos efectos en términos de la actitud de los alumnos hacia la asignatura, su calificación en el examen, y su opinión sobre las animaciones.

El contexto de esta evaluación es la asignatura de *Bases de Lenguajes de Programación*, impartida durante el primer año de las titulaciones de *Ingeniería Técnica en Informática de Gestión* e *Ingeniería Técnica en Informática de Sistemas*, de la *Universidad Rey Juan Carlos*. La evaluación se centró en la segunda mitad de la asignatura, que está dedicada al paradigma de programación funcional, donde se utiliza el lenguaje Hope.

En esta sección describiremos el diseño y resultados de esta evaluación. La documentación utilizada durante la evaluación, así como los detalles de los análisis estadísticos realizados se encuentran en el apéndice D.

### 5.5.1. Participantes

Los participantes son 132 estudiantes matriculados en la asignatura antes mencionada. Los dividimos en tres grupos: los que siguen un enfoque típico de enseñanza de la asignatura sin utilizar animaciones, que llamaremos GT ($n$=42); los que usan las animaciones sólo para verlas, que llamaremos GV ($n$=50); y los que se dedican a construir sus propias animaciones, que llamaremos GC ($n$=40).

La participación de los estudiantes es voluntaria e incentivada. El incentivo consiste en un máximo de 0,25 puntos extra en la nota final. Hemos filtrado aquellos estudiantes repetidores; de esta forma ninguno de los participantes tiene conocimientos previos de programación funcional.

### 5.5.2. Variables

La variable independiente de la evaluación es el nivel de implicación con las animaciones. Las variables dependientes son la eficacia pedagógica, medida con las respuestas al examen de la asignatura sobre el paradigma funcional (véase apéndice D.1); la actitud de los estudiantes hacia la asignatura y las animaciones, medida con los datos de presentación de los alumnos al examen y las consultas de las animaciones disponibles en la página Web de la asignatura; y la opinión de los estudiantes, recogida mediante dos cuestionarios (uno para GV y otro para GC), ambos disponibles en los apéndices D.2.1 y D.2.2 respectivamente.

### 5.5.3. Protocolo

Esta evaluación se ha realizado con vistas a analizar los efectos del uso de las animaciones a largo plazo. La idea principal es que los alumnos se familiaricen con una forma de usar las animaciones, y las utilicen durante todo el desarrollo de la materia. En primer lugar describiremos la estructura de la materia, después explicaremos la metodología didáctica empleada con cada grupo, y finalmente detallaremos el desarrollo temporal de la evaluación.

**Estructuración de la materia**

La materia pretende dar una visión básica del paradigma de programación funcional, sin entrar en profundidad con características avanzadas como funciones de orden superior, o la estrategia de evaluación perezosa. El lenguaje funcional utilizado es Hope [171]. En este lenguaje se usa la estrategia de evaluación impaciente, más fácil de entender; otros lenguajes como Haskell [104] usan la estrategia de evaluación perezosa. En Hope no se mezclan características de otros paradigmas, como hace Ml [138] con el paradigma imperativo, o Scheme [181] y Lisp [166] con el lógico. Finalmente, Hope tiene una sintaxis más literal, al contrario que los mencionados Scheme o Lisp. La materia se divide en ocho unidades temáticas:

1. El paradigma funcional, donde se exponen los conceptos básicos y las propiedades del paradigma, sin entrar en detalles del lenguaje utilizado.

2. Características básicas del lenguaje Hope, donde se explican las propiedades básicas del lenguaje: operadores predefinidos, precedencia y asociatividad, tipos básicos de datos, y expresiones sencillas.

3. Conceptos básicos de recursividad. El mecanismo de repetición de procesos en Hope es la recursividad, por ello tenemos un tema enteramente dedicado a ella.

4. Operadores prefijos e infijos. Por defecto, las funciones definidas por el programador se usan de forma prefija, mientras que los operadores "básicos" del lenguaje, como los aritméticos, los lógicos, o los de comparación, se usan de forma infija. Sin embargo, en Hope se pueden definir funciones para usar de forma infija. Además se hace hincapié en la definición de la precedencia de operadores.

5. Tipos de datos simples definidos por el usuario. Declaración y uso de tipos de datos simples creados por el programador. Son simples, puesto que sus valores se definen por extensión (p.ej. tipos enumerados), o se definen en función de otros tipos declarados previamente, como los básicos.

6. Definiciones locales. Declaración y ejecución de definiciones locales. Optimización por medio de declaraciones locales.

7. Tipos de datos recursivos. Declaración y uso de tipos de datos recursivos creados por el programador. Son recursivos ya que su dominio se puede definir de forma inductiva. Tipos recursivos estándar, como listas o árboles binarios.

8. Polimorfismo. Ventajas de los tipos polimórficos, variables de tipo y uso de listas polimórficas.

**Metodología didáctica utilizada**

Los estudiantes de los tres grupos recibieron tanto clases magistrales como sesiones de laboratorio; en estas últimas se les proponían un conjunto de prácticas optativas sobre los conceptos explicados anteriormente. La figura 5.14 muestra un esquema global de las metodologías usadas en la evaluación. Aunque los profesores eran distintos –GC y GV tuvieron al mismo profesor, mientras que GT tuvo a otro distinto–, el contenido de los materiales didácticos utilizados era común: teoría en transparencias y ejercicios de un libro [238] editado expresamente para la asignatura.

<table>
<tr><th>Grupo / Clases</th><th>Teóricas</th><th>Laboratorios</th></tr>
<tr><td>GT</td><td>Exposición + Ejemplos</td><td>Ejercicios</td></tr>
<tr><td>GV</td><td rowspan="2">Exposición + Ejemplos + Animaciones</td><td>Ver animaciones + Ejercicios</td></tr>
<tr><td>GC</td><td>Construir animaciones + Ejercicios</td></tr>
</table>

Figura 5.14: Esquema explicativo de las distintas metodologías didácticas utilizadas en cada grupo

Las clases magistrales recibidas por los estudiantes del GT seguían una metodología docente típica con utilización de transparencias (PowerPoint) y pizarra. Las sesiones de laboratorio consistían en la codificación de problemas propuestos por el profesor, utilizando el entorno WinHIPE, y la posterior explicación de la solución.

Las clases magistrales recibidas por los estudiantes de GV y GC seguían una metodología docente similar a GT, junto con la exposición de animaciones por parte del profesor. Dichas animaciones estaban hechas con WinHIPE[2] y eran explicadas por el profesor durante su exposición. Las sesiones de laboratorio de ambos grupos eran totalmente distintas.

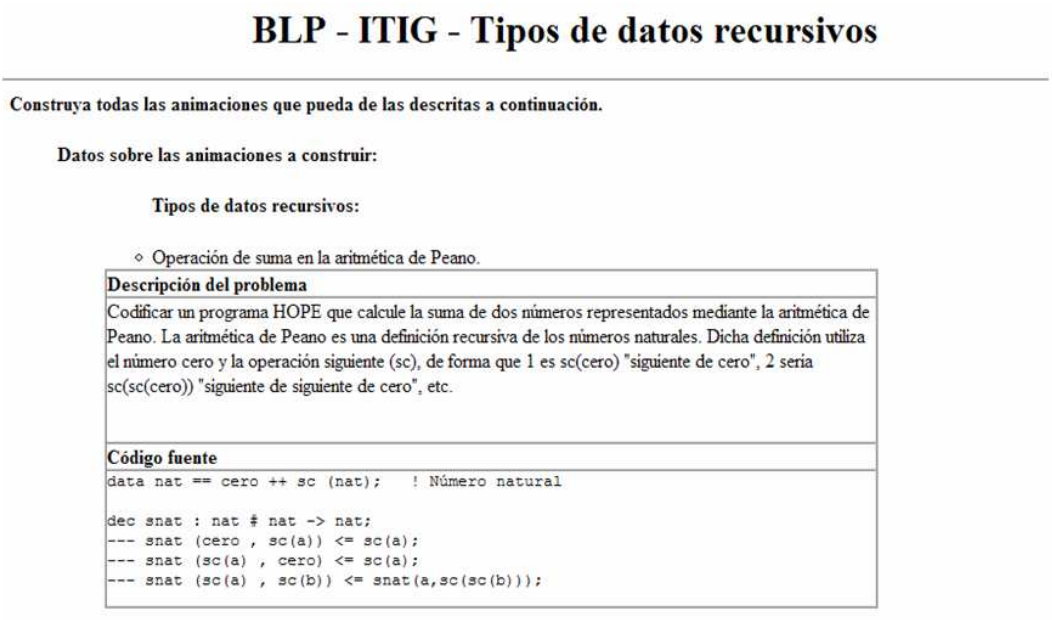
Las sesiones de laboratorio del GV consistían en el estudio de un conjunto de animaciones generadas con WinHIPE; la figura 5.15 muestra un ejemplo de enunciado. Dichas animaciones implementaban problemas del libro de ejercicios antes mencionado.


Figura 5.15: Ejemplo de enunciado de sesión de prácticas del GV. Se proporcionaba a los alumnos un conjunto de animaciones para su estudio. Se pueden ver todos los enunciados en la siguiente dirección: `http://www.escet.urjc.es/~jurquiza/Tesis/aniweb.html`

Las sesiones de laboratorio del GC consistían en la construcción de un conjunto de animaciones con WinHIPE; la figura 5.16 muestra un ejemplo de enunciado. Dichas animaciones implementaban

[2]La flexibilidad de configuración de la apariencia gráfica de las visualizaciones, nos permitió generar animaciones con un tamaño de letra mucho más grande, conveniente para su uso en clase.

problemas del libro de ejercicios antes mencionado. A los estudiantes se les proporcionaba la descripción del problema y el código fuente de la solución; al tener que generar la animación, ellos tenían que escribir la descripción de la solución y generar las visualizaciones correspondientes.



**BLP - ITIG - Tipos de datos recursivos**

**Construya todas las animaciones que pueda de las descritas a continuación.**

**Datos sobre las animaciones a construir:**

**Tipos de datos recursivos:**

- Operación de suma en la aritmética de Peano.

**Descripción del problema**

Codificar un programa HOPE que calcule la suma de dos números representados mediante la aritmética de Peano. La aritmética de Peano es una definición recursiva de los números naturales. Dicha definición utiliza el número cero y la operación siguiente (sc), de forma que 1 es sc(cero) "siguiente de cero", 2 sería sc(sc(cero)) "siguiente de siguiente de cero", etc.

**Código fuente**

```
data nat == cero ++ sc (nat);   ! Número natural

dec snat : nat # nat -> nat;
--- snat (cero , sc(a)) <= sc(a);
--- snat (sc(a) , cero) <= sc(a);
--- snat (sc(a) , sc(b)) <= snat(a,sc(sc(b)));
```

Figura 5.16: Ejemplo de enunciado de sesión de prácticas del GC. Se proporcionaba a los estudiantes la descripción de un problema y el código fuente que lo resolvía, teniendo que construir la animación. Se pueden ver todos los enunciados en la siguiente dirección: `http://www.escet.urjc.es/~jurquiza/Tesis/aniweb.html`

### Desarrollo temporal de la evaluación

Durante la primera unidad no hubo sesión de laboratorio. Las sesiones de laboratorios correspondientes a la segunda y tercera unidades se dedicaron a familiarizar a los estudiantes con el entorno de programación WinHIPE, así como con el lenguaje de programación Hope.

A partir de aquí, y durante las siguientes cuatro unidades empezamos a aplicar los diferentes tratamientos a cada grupo en sus respectivas sesiones de laboratorio. Al finalizar cada sesión de laboratorio, los estudiantes tenían que responder a un test de conocimientos[3] sobre los conceptos que habían estado practicando. Algunos de los ejercicios propuestos en este test implicaban el uso del entorno para codificar programas.

En la última sesión de tratamiento se facilitó a los estudiantes un cuestionario de opinión sobre el uso de las animaciones como herramientas educativas. La última unidad no tuvo un tratamiento diferenciado entre los grupos, y la última sesión de laboratorio del curso se dedicó a la resolución de dudas de los estudiantes. Una semana después se celebró el examen de la asignatura, de donde obtuvimos los datos para medir el conocimiento de los estudiantes sobre la materia.

[3] Véase: `http://www.escet.urjc.es/~jurquiza/Tesis/aniweb.html`

### 5.5.4. Resultados de la evaluación

A continuación exponemos los resultados de la evaluación según los tres aspectos antes mencionados: actitud de los estudiantes respecto a la asignatura y las animaciones, eficacia pedagógica de los tres enfoques, y opinión subjetiva de los estudiantes. Los detalles del estudio estadístico se encuentran en el apéndice D.3.

#### Actitud de los estudiantes

Hemos medido la actitud de los estudiantes en función de: la presentación al examen de junio de la asignatura, y el uso real de las animaciones publicadas en las páginas Web de la asignatura[4].

En la tabla 5.1 mostramos los datos de presentación al examen por parte de los alumnos de la asignatura. Como se puede ver, los datos de presentación al examen de todos los estudiantes matriculados son muy similares en los tres grupos, e independientes del grupo al que pertenecen según el test de $\chi^2$ *de Pearson*. Sin embargo, las diferencias aparecen cuando únicamente estudiamos los alumnos que participaron en la evaluación. El test de $\chi^2$ *de Pearson* detecta dependencias prácticamente significativas ($p < 0,058$), lo que nos indica que es posible la existencia de diferencias entre los grupos, pero no entre todos; de hecho, podemos observar que el grupo GC se ha presentado en mayor medida que los otros dos.

| | Matriculados en el grupo | Participantes en la evaluación |
|---|---|---|
| Estudiantes GC | 65,5 % (78/119) | 83,93 % (47/56) |
| Estudiantes GV | 60,8 % (73/120) | 69,64 % (39/56) |
| Estudiantes GT | 60,4 % (90/149) | 64,81 % (35/54) |
| Test de dependencia | $\chi^2(2, 388) = 0,865, p = 0,649$ | $\chi^2(2, 167) = 5,73, p = 0,057$ |

Tabla 5.1: Datos de presentación al examen. Analizamos en cada grupo el porcentaje de estudiantes presentados al examen de la asignatura, respecto de todos los estudiantes matriculados en la asignatura, o sólo aquellos que hayan participado en la evaluación. Detalles disponibles en el apéndice D.3.1

Para profundizar en el estudio de la presentación de los estudiantes al examen de la asignatura, hemos analizado los datos de los grupos por pares; mostramos los resultados en la tabla 5.2.

| | Diferencia en % de presentación | Test de diferencias significativas |
|---|---|---|
| GC vs GT | 19,12 % (GC>GT) | $U = 1223, p =$ **0,022** |
| GC vs GV | 14,29 % (GC>GV) | $U = 1348,5, p = 0,054$ |
| GV vs GT | 4,83 % (GV>GT) | $U = 1348,5, p = 0,688$ |

Tabla 5.2: Análisis de presentación al examen por parejas utilizando el test $U$ de Mann-Whitney. Analizamos diferencias significativas entre los grupos contando únicamente con los participantes en la evaluación. Detalles disponibles en el apéndice D.3.1

Se puede ver que sí existen diferencias significativas entre GC y GT, el porcentaje de presentación de GC fue un 19,12 % mayor que el de GT; así como diferencias prácticamente significativas entre GC

[4] http://www.escet.urjc.es/~jurquiza/BLPCol0506/blp0506.htm

y GV, donde el porcentaje de presentación de GC fue un 14,29 % mayor que el de GV; finalmente, no encontramos diferencias significativas entre GV y GT.

En cuanto al uso real de las animaciones, monitorizamos los accesos realizados a estas durante la semana anterior al examen de la asignatura, la cantidad de accesos fue muy numerosa, 542 accesos en total.

**Eficacia pedagógica**

Hemos medido la eficacia pedagógica con las calificaciones de los estudiantes en el examen de la asignatura. Sólo hemos considerado a los participantes en la evaluación no repetidores; de esta forma eliminamos la posibilidad de que algunos estudiantes tengan conocimientos previos.

El paradigma funcional representa la mitad de esta asignatura, por ello sólo consideraremos los resultados obtenidos en las preguntas y ejercicios relacionados con el paradigma funcional (véase apéndice D.1). Todas las calificaciones están normalizadas al rango [0,1]. Para valorar la calificación global en el paradigma funcional, utilizaremos los mismos pesos que se asignan en el cómputo de la nota final de la asignatura. En la calificación global se contaba la parte práctica obligatoria ($O$), cuyo peso es de 25 % de la nota, y la parte teórica cuyo peso es de 75 %; esta a su vez se componía de una cuestión teórica ($C$) cuyo peso es de 33,3 % y un problema ($P$), cuyo peso es el 66,6 % restante de la nota del examen. La fórmula de cálculo para saber la nota final de los alumnos en el paradigma funcional es:

$$\text{Calificación total } T(O, C, P) = 0,25 * O + 0,75 * ((C + 2 * P)/3)$$

En la tabla 5.3 mostramos el análisis de las calificaciones por cada ejercicio junto con la calificación total, de los tres grupos. Se puede observar que hay dependencias prácticamente significativas en $C$ y $T$. Además, podemos percibir un cierto patrón en estos resultados; GC y GV obtienen calificaciones similares entre sí, y mejores que GT. Este mismo patrón también aparece en $P$, aunque menos significativo ($p = 0,141$); así que haremos un análisis más detallado por pares de los tres $C$, $P$ y $T$.

| | GC | GV | GT | Análisis de dependencia |
|---|---|---|---|---|
| Práctica obligatoria($O$) | 0,5486 | 0,65 | 0,525 | $\chi^2(2, 116) = 1,77, p = 0,412$ |
| Cuestión teórica ($C$) | 0,7122 | 0,7233 | 0,5194 | $\chi^2(2, 116) = 5,74, p = 0,057$ |
| Problema ($P$) | 0,3954 | 0,3707 | 0,2808 | $\chi^2(2, 116) = 3,92, p = 0,141$ |
| Calificación total ($T$) | 0,5129 | 0,5287 | 0,4015 | $F(2, 113) = 2,528, p = 0,084$ |

Tabla 5.3: Datos de calificaciones obtenidas en el examen por los participantes no repetidores. Analizamos, por cada ejercicio, la nota media de cada grupo y el análisis de dependencia asociado. $O$, $C$ y $P$ no son normales por lo que usaremos el test de Kruskal-Wallis; mientras que $T$ sí es normal, por lo que usaremos un análisis ANOVA. Detalles disponibles en el apéndice D.3.2

Analizando los resultados de la cuestión teórica, el problema y la calificación total por pares (véase tabla 5.4), descubrimos que sí existen diferencias significativas entre GV y GT; GV mejoró los resultados de GT un 20,4 % en la cuestión teórica y un 12,7 % en la calificación total. También existen diferencias casi-significativas ($p < 0,075$) entre GC y GT; GC mejoró los resultados de GT un 19,3 % en la cuestión teórica, un 11,5 % en el problema y un 11,1 % en la calificación total. Sin embargo, entre GC y GV no hay diferencias significativas.

Finalmente, hemos hecho un análisis de las calificaciones cualitativas en función de la calificación total. Las calificaciones cualitativas serían: suspenso, aprobado, notable y sobresaliente. Detectamos

| | GC-GV | GC-GT | GV-GT |
|---|---|---|---|
| $C$ | $U = 765, p = 0,743$ | $U = 505, p = 0,060$ | $U = 565,5, p =$ **0,028** |
| $P$ | $U = 743,5, p = 0,615$ | $U = 492,5, p = 0,052$ | $U = 630,5, p = 0,153$ |
| $T$ | $t(78) = -0,268, p = 0,789$ | $t(71) = 1,813, p = 0,074$ | $t(77) = 2,028, p =$ **0,046** |

Tabla 5.4: Análisis de calificaciones en el examen por parejas. Analizamos diferencias entre significativas, utilizado los test U de Mann-Whitney, y t-Student, entre los grupos, contando únicamente con los participantes en la evaluación. Destacamos en negrita las diferencias significativas. Detalles disponibles en el apéndice D.3.2

que las calificaciones dependen significativamente de los grupos, $\chi^2(6, 116) = 13,408, p =$ **0,037**. En la tabla 5.5 mostramos los rangos de calificación cuantitativa correspondientes, así como los porcentajes para cada grupo. Como se puede ver, el grupo GT es el que más suspensos tiene, el GC aprobados y notables, y el GV el que más sobresalientes tiene.

| | Rango | GC | GV | GT |
|---|---|---|---|---|
| Suspenso | x <0,5 | 43,2 % | 46,5 % | **61,1 %** |
| Aprobado | 0,5 <= x <0,7 | **35,1 %** | 25,6 % | 22,2 % |
| Notable | 0,7 <= x <0,9 | **21,6 %** | 14,0 % | 16,7 % |
| Sobresaliente | x >0,9 | 0,0 % | **14,0 %** | 0,0 % |

Tabla 5.5: Análisis de calificaciones cualitativas. Para los cuatro niveles de calificación mostramos el rango asociado de las calificaciones cuantitativas, así como el porcentaje de estudiantes que obtuvieron dicha calificación en cada uno de los tres grupos, en negrita destacamos el grupo que obtuvo mayor porcentaje en cada calificación. Detalles disponibles en el apéndice D.3.2

**Opinión de los estudiantes**

Hemos recogido la opinión subjetiva de los estudiantes sobre las animaciones con un cuestionario (véase apéndice D.2). Sólo trabajaremos con los grupos GC y GV, ya que GT no tuvo ningún contacto con ellas. En este cuestionario hicimos cuatro preguntas sobre la experiencia de los estudiantes con las animaciones; en la tabla 5.6 resumimos sus resultados. Podemos apreciar que la opinión de los estudiantes es muy buena. El 93 % piensan que las animaciones ayudan a la comprensión de los conceptos, con una ligera diferencia entre GC y GV. Además, existe total unanimidad en la facilidad de construcción y uso de las animaciones. Y el 91,5 % de los estudiantes piensan que las animaciones son útiles, de nuevo con una ligera diferencia entre GC y GV. Finalmente les preguntamos sobre la disponibilidad de distintos formatos para ver las animaciones, el 78,9 % mostraron su acuerdo en la utilidad de esta característica.

También les preguntamos sobre el modo de uso de las animaciones que más favorecería el aprendizaje; les propusimos cuatro opciones:

- Construir es mejor que ver.
- Ver es mejor que construir.
- Ambas son iguales pero deberían usarse conjuntamente.

| | GC | GV | Total |
|---|---|---|---|
| Ayuda a la comprensión de los conceptos | 86,2 % | 100 % | 93 % |
| Facilidad de construcción | 100 % | NA | 100 % |
| Facilidad de uso de las animaciones | NA | 100 % | 100 % |
| Utilidad de las animaciones | 83,4 % | 100 % | 91,5 % |
| Utilidad sobre la disponibilidad en varios formatos | 83,3 % | 74,3 % | 78,9 % |

Tabla 5.6: Opinión de los alumnos sobre su experiencia con las animaciones. Cada fila se refiere a un aspecto. Las columnas indican, por cada grupo y en general, el % de estudiantes que están de acuerdo o totalmente de acuerdo en valorar positivamente el aspecto sobre el que se pregunta. *NA* significa no aplicable, ya que al GV no se le preguntó por la construcción de animaciones, como al GC no se le preguntó por la visión

- Ambas son iguales, cualquiera vale.

En la figura 5.17 mostramos un resumen de las respuestas en cada grupo. Como se puede observar, la importancia asignada por los estudiantes al proceso de construcción de las animaciones depende claramente del grupo: los estudiantes de GC la dan mayor importancia, los de GV la dan la misma que a la visión de animaciones. Aún así, un 89 % de los estudiantes del GC y un 80 % de los estudiantes del GV, usaría la construcción de animaciones, sola o junto con la visión.

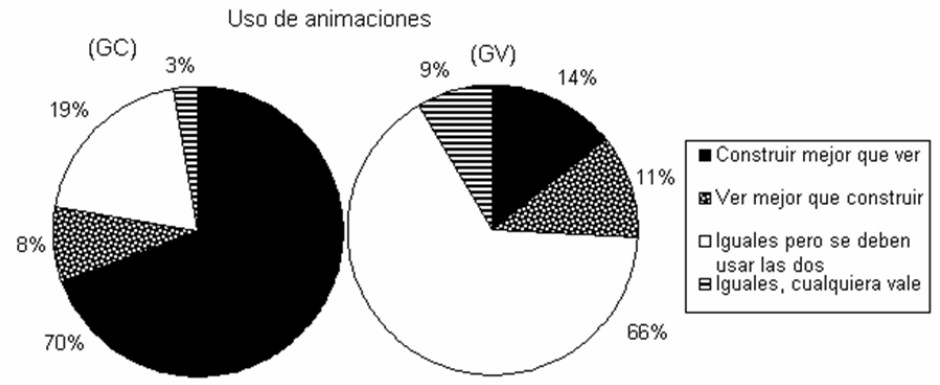



Figura 5.17: Opinión de los estudiantes de los grupos GC y GV sobre el modo de uso de las animaciones, ver o construir, ¿qué favorece más el aprendizaje?

### 5.5.5. Generalización de los resultados

En esta evaluación hemos corregido las debilidades de la anterior, ya que los tratamientos han tenido una duración más amplia, han cubierto gran parte de la materia explicada, y se han aplicado a un número extenso de estudiantes. Por lo tanto, la representatividad de los resultados es mayor. Para conseguir resultados más generalizables debemos asegurarnos de que ningún factor intermedio, ya sea interno o externo a la evaluación, pueda afectar a los resultados. A continuación analizamos ambas posibilidades.

**Análisis de factores intermedios *internos* a la evaluación**

Uno de los puntos débiles de la evaluación a corto plazo fue la existencia de dos factores: el nivel de implicación y el tiempo dedicado al estudio. En esta evaluación nos hemos asegurado de que todos los estudiantes trabajen el mismo tiempo en las distintas metodologías, tratando de dejar una única variable independiente real, el nivel de implicación con las animaciones.

Sin embargo existe una diferencia entre los grupos que no hemos controlado, los profesores encargados de cada grupo. Aunque hemos minimizado el posible impacto de esta diferencia utilizando los mismos materiales en común –transparencias y libro de ejercicios–, analizaremos la existencia de dependencias entre los profesores y los resultados del experimento. Como ya dijimos anteriormente, GC y GV tuvieron al mismo profesor (profesor A), mientras que GT tuvo otro distinto (profesor B).

Es evidente que los datos sobre la presentación de los estudiantes al examen son independientes de los profesores. GC y GV tuvieron al mismo profesor pero GC se ha presentado en mucha mayor medida que GV. Además, GV y GT tuvieron distintos profesores, pero el porcentaje de presentados fue similar. Por lo tanto la motivación de los alumnos es independiente del profesor asignado a los grupos.

Los resultados obtenidos en las calificaciones sí podrían estar relacionados con los profesores, ya que GV y GC (profesor A) obtuvieron mejores calificaciones que GT (profesor B). La participación en la evaluación era voluntaria; por lo tanto no todos los estudiantes que asistían a clase y se presentaron al examen participaron en ella. Si las calificaciones dependen de los profesores, y no del tratamiento recibido durante los laboratorios, debería existir una correlación fuerte entre el profesor y la calificación obtenida por los alumnos, participantes o no en la evaluación. Sin embargo el análisis de correlación de la tabla 5.7 muestra que las calificaciones obtenidas por los alumnos son independientes de los profesores.

| | Análisis de correlación |
|---|---|
| Práctica obligatoria | $\chi^2(3, 165) = 0,651, p = 0,885$ |
| Cuestión teórica | $\chi^2(3, 165) = 3,186, p = 0,364$ |
| Problema | $\chi^2(3, 165) = 0,403, p = 0,940$ |
| Calificación total | $\chi^2(3, 165) = 4,125, p = 0,248$ |

Tabla 5.7: Análisis de correlación entre los profesores y las calificaciones de los estudiantes. Los detalles de este análisis se encuentran en el anexo D.3.3

**Análisis de factores intermedios *externos* a la evaluación**

La situación ideal de esta evaluación sería un entorno aséptico donde el único factor influyente fuera el grado de implicación de los estudiantes con las animaciones. Sin embargo, los estudiantes son personas individuales con personalidad propia, sobre los que pueden afectar condicionantes externos al tratamiento, como la preparación académica previa a la universidad, el hábito de estudio, o la actitud hacia la titulación que están estudiando.

Para tratar de aislar al máximo el factor, detectando la posible influencia de estos condicionantes externos, hemos estudiado los datos de una asignatura del mismo curso y examinada en la misma convocatoria. Esta asignatura es *Cálculo*, y en ella no se ha utilizado ningún tratamiento relacionado con nuestra evaluación; por lo tanto la podemos utilizar como asignatura de control. En concreto,

hemos verificado si existe alguna relación entre las calificaciones obtenidas por los alumnos en esta asignatura, con respecto a la asignatura utilizada en nuestra evaluación.

Las fórmulas de cálculo de las calificaciones numéricas varían entre las asignaturas, por lo tanto no haremos ningún análisis al nivel de las calificaciones cuantitativas. El análisis de dependencia lo haremos con las calificaciones cualitativas desde dos puntos de vista: con las cinco calificaciones posibles –no presentado, suspenso, aprobado, notable o sobresaliente– y con la calificación como no presentado, suspenso o no suspenso (aprobado, notable y sobresaliente). Mostramos los resultados de ambos análisis en la tabla 5.8.

| | Calificaciones cualitativas | No presentados - aprobados - suspensos |
|---|---|---|
| GC | $(N = 24; \tau - b = 0,096; p = 0,601)$ | $(N = 24; \tau - b = 0,125; p = 0,493)$ |
| GV | $(N = 34; \tau - b = 0,255; p = 0,081)$ | $(N = 34; \tau - b = 0,122; p = 0,477)$ |
| GT | $(N = 36; \tau - b = 0,123; p = 0,413)$ | $(N = 36; \tau - b = 0,051; p = 0,733)$ |

Tabla 5.8: Análisis de correlación entre la asignatura de la evaluación, programación funcional, y la de control, Cálculo. Los detalles de este análisis se encuentran en el anexo D.3.4

No existe ninguna correlación significativa. Aunque calificaciones cualitativas de GV se acercan a un nivel significativo (p=0,081), la magnitud de la correlación es muy baja $\tau - b < 0,26$. Según estos resultados, podemos asegurar que los datos recogidos en la asignatura evaluada no tienen correlación alguna con los datos de la asignatura de control, donde no se han utilizado animaciones. Por lo tanto, las calificaciones obtenidas en programación funcional se deben a los tratamientos realizados en dicha asignatura, y no a otras causas cuyo efecto podría verse en otras asignaturas, como el hecho de ser mejor o peor estudiante.

### 5.5.6. Interpretación de los resultados

Los diferentes niveles de implicación requieren del estudiante diferentes formas de interactuar con las animaciones y con lo que representan, ejecución de programas funcionales. Hundhausen et al. [94] hicieron un meta-estudio sobre la eficacia de las animaciones de algoritmos, y detectaron que la teoría pedagógica que mejor podía predecir la eficacia pedagógica era el constructivismo; cuanto mayor es la implicación de los estudiantes con las animaciones mayor es el aprendizaje.

La implicación de los estudiantes que construyen animaciones de programas, es mucho mayor que aquellos que solamente las ven. Construir una animación implica un conocimiento más profundo del programa ejecutado, qué pasos son los importantes en cada momento de la ejecución, de forma que la animación se suficientemente explicativa de la ejecución del programa. Además, los estudiantes que construyeron animaciones tenían que escribir una descripción de la solución al problema implementada en el programa que estaban animando, lo que implica una tarea de reflexión sobre el programa. Por otro lado, la visión de las animaciones requiere que los estudiantes sean capaces de interpretar el funcionamiento del programa, sabiendo qué pasos se han ejecutado para llegar a cada fotograma de la animación. En general la construcción de la animación requiere de un estudio más detenido del programa a animar, consiguiendo que el estudiante dedique más tiempo y atención al programa. Viendo animaciones, el tiempo y grado de atención dedicado a estas es menor, ya que a los estudiantes les basta con interpretar para ellos mismos lo que están viendo, no tienen que explicarlo posteriormente a nadie. También podríamos decir que la construcción de animaciones obliga a reflexionar más que la

simple visión de estas.

Claramente, el nivel de implicación es mayor para los estudiantes que construyeron animaciones, que para los que sólo las vieron. Por lo tanto, los primeros deberían obtener mejores resultados que los segundos. Sin embargo, a primera vista esto no es así. No hemos encontrado diferencias significativas en las calificaciones entre los grupos GC y GV, pero sí en el porcentaje de presentados al examen. Vamos a calcular cuántos aprobados más hay en GC por el hecho de haberse presentado en un 14,3 % más que GV. Repartiendo esta diferencia de presentados según los suspensos y aprobados de GC (43,2 % y 56,7 % respectivamente), y teniendo en cuenta que hay más presentados en GC, tenemos que existe un 8,1 % más de aprobados debido a la construcción de animaciones.

Además, la presentación al examen es un indicador de la actitud de los estudiantes hacia la asignatura. La construcción de animaciones ha influido claramente en esta actitud. En concreto, la construcción de animaciones (83,9 % de presentados) aumenta la presentación al examen con respecto a la simple visión de las mismas (69,6 % de presentados).

Finalmente, comparando los resultados con el grupo de control que no usó animaciones de ninguna forma, comprobamos que estas han tenido un efecto positivo en GC y GV en conjunto. Los grupos que utilizaron animaciones han obtenido mejores resultados, un 11,9 % más en la nota cuantitativa, que los que no utilizaron animaciones. En lo que se refiere a las calificaciones cualitativas, la diferencia principal se encuentra en la proporción suspensos/aprobados (incluyendo a los notables y sobresalientes junto con los aprobados), donde los grupos que usaron animaciones obtuvieron un 16,3 % más de aprobados que el grupo que no usó animaciones. Además, los estudiantes que no utilizaron animaciones fueron los que menos se presentaron al examen (sólo un 64,8 % de presentados).

## 5.6. Resumen de aportaciones

En este capítulo hemos descrito las modificaciones realizadas sobre la composición de las animaciones con respecto a su versión anterior, y las implicaciones en el proceso de construcción. También hemos detallado su evaluación como herramientas pedagógicas, tanto desde el punto de vista clásico –material de consulta–, como desde un punto de vista constructivista –experiencia de aprendizaje–. Así pues, podemos clasificar las aportaciones en dos grupos: tecnológicas y educativas.

Desde el punto de vista *tecnológico*, hemos integrado en el entorno de programación la construcción de animaciones con explicaciones textuales añadidas. Además tanto el diseño como la estructuración de los contenidos de las animaciones Web permiten que estas sean reutilizables. Esta reutilización se da en dos niveles: a nivel de plataforma, ya que se pueden consultar por medio de cualquier navegador Web y gestionar con cualquier aplicación capaz de manejar documentos XML; y a nivel de usuario, ya que la propia animación contiene toda la información necesaria para su posterior mantenimiento o modificación. Finalmente, aportamos un modelo que permite manejar colecciones de animaciones Web de forma sencilla. En las colecciones, las animaciones son documentos cuya complejidad interna es transparente al usuario, permitiendo realizar tareas típicas –creación, modificación, eliminación, o publicación– de forma sencilla.

Desde el punto de vista *educativo*, aportamos evidencia empírica de los beneficios que presenta usar la construcción de animaciones como recurso pedagógico frente al enfoque típico de la clase expositiva y el laboratorio de resolución de problemas. También aportamos evidencia empírica de los beneficios que representa la consulta de las animaciones con explicaciones textuales añadidas respecto al enfoque típico anteriormente mencionado. Sorprendentemente, la construcción de animaciones y su consulta aportan

los mismos beneficios en términos de resultados académicos, aunque no en lo que a motivación hacia la asignatura se refiere. En el capítulo de conclusiones ahondaremos sobre este resultado. Finalmente, aportamos el material didáctico utilizado para impartir las clases de programación funcional, que tan buen resultado ha ofrecido.

# Capítulo 6

# Conclusiones y trabajos futuros

En esta tesis hemos presentado un modelo mejorado de generación semiautomática de animaciones de programas funcionales para su uso en ámbitos educativos. La ejecución de los programas pertenecientes al paradigma funcional se puede modelar como un proceso de reescritura de expresiones. Así, el historial de expresiones intermedias producidas durante este proceso sirve como representación de la ejecución de los programas. El modelo de animación presentado en esta tesis construye las animaciones de programas funcionales a partir de las visualizaciones estáticas discretas de cada expresión intermedia, que son la representación de los distintos estados por los que ha pasado la ejecución del programa.

Las dos primeras palabras del título de esta tesis son *Generación Semiautomática*. Por lo tanto, parte de las tareas estarán automatizadas y parte necesitarán de la intervención del usuario. La parte automática consiste en la producción de todas las expresiones intermedias de la ejecución y sus representaciones gráficas, así como de la generación de las animaciones en formato Web. La parte manual consiste la adaptación de ciertos aspectos de la representación gráfica de las expresiones, la selección de las visualizaciones estáticas que formarán la animación, y la integración de explicaciones textuales en las animaciones en formato Web.

Las dos últimas palabras del título de esta tesis son *Fines Educativos*. Por lo tanto, las animaciones generadas no son meras representaciones gráficas de la ejecución de programas funcionales. Hemos diseñado las animaciones con el objetivo de que sean herramientas pedagógicas eficaces, por ello hemos tenido que cambiar sus contenidos, apariencia y proceso de construcción. Pero no nos hemos conformado con el producto final del proceso de generación, las animaciones, sino que hemos diseñado el propio proceso de generación como una tarea educativa en sí misma. De esta forma, al ser un proceso sencillo, el estudiante es capaz de concentrar sus esfuerzos en diseñar una animación que sea suficientemente representativa de la ejecución del programa, lo que le permite aprender mejor los conceptos utilizados en el programa.

Los trabajos realizados durante el desarrollo de esta tesis han perseguido este objetivo: que tanto las animaciones como su propio proceso de construcción sean herramientas pedagógicas eficaces. Para ello, hemos puesto especial atención en conseguir que la generación de las animaciones se realice con el menor esfuerzo posible, centrándonos en aspectos puramente interactivos. Una vez que nos hemos asegurado de que el proceso básico de generación de animaciones es sencillo, dedicamos nuestros esfuerzos al diseño de las animaciones como herramientas pedagógicas eficaces. Ahora, los puntos clave fueron: la usabilidad de las animaciones, su reutilización a nivel de usuario y plataforma, la integración con explicaciones

textuales, y la valoración de su eficacia pedagógica. Como resultado de todos estos trabajos, tanto el proceso de construcción de animaciones, como el propio entorno WinHIPE han mejorado notablemente.

Expondremos nuestras conclusiones en el mismo orden en que se ha desarrollado nuestro trabajo, primero los aspectos interactivos del proceso de generación de animaciones, y después los aspectos pedagógicos de las animaciones y su proceso de construcción.

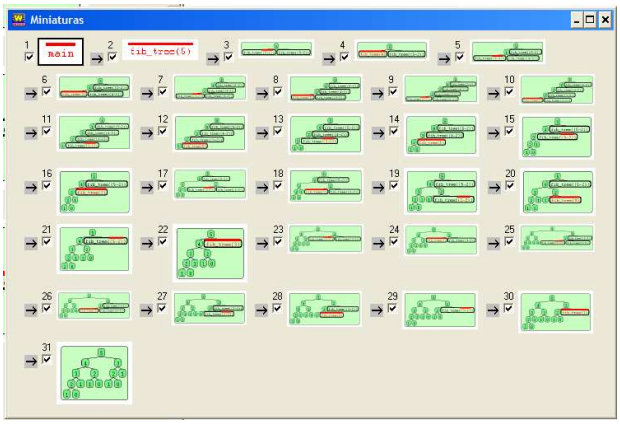
## 6.1. Aspectos interactivos

Antes de la realización de esta tesis, y desde el punto de vista de la interacción básica del usuario, el procedimiento de construcción de animaciones diseñado en WinHIPE era bastante simple, ya que sólo consistía en la selección de las visualizaciones estáticas que formarían parte de la animación. Además, WinHIPE trataba de ofrecer una visión global de la ejecución con las miniaturas, versiones reducidas de las visualizaciones estáticas. Sin embargo, la implementación real presentaba algunos problemas que obligaban al usuario a dedicar demasiado esfuerzo a la selección de las visualizaciones, haciendo inútil la simplicidad del propio proceso.

Tras un estudio de la interfaz existente para la selección de las visualizaciones estáticas, hemos comprobado que no se facilita la compresión de la ejecución de los programas, tanto a nivel global como de pasos de ejecución individuales. El origen de estos problemas está en dos características interactivas que entorpecen dicho proceso de selección: la calidad de las miniaturas, y la integración de vistas globales y más detalladas de la ejecución.

En primer lugar nos hemos centrado en aumentar la calidad de la miniaturas individualmente. Hemos mejorado la representación de las expresiones intermedias, ayudando a reconocer el punto de ejecución actual (redex) mediante técnicas de *pretty-printing*. También hemos mejorado la representación de las miniaturas de dos formas distintas: manteniendo las proporciones de las visualizaciones originales, y usando un algoritmo de reducción de imágenes que proporciona alta calidad en un tiempo razonable. La figura 6.1 muestra un ejemplo comparativo del efecto de estas mejoras en las miniaturas.


a) Miniaturas antiguas

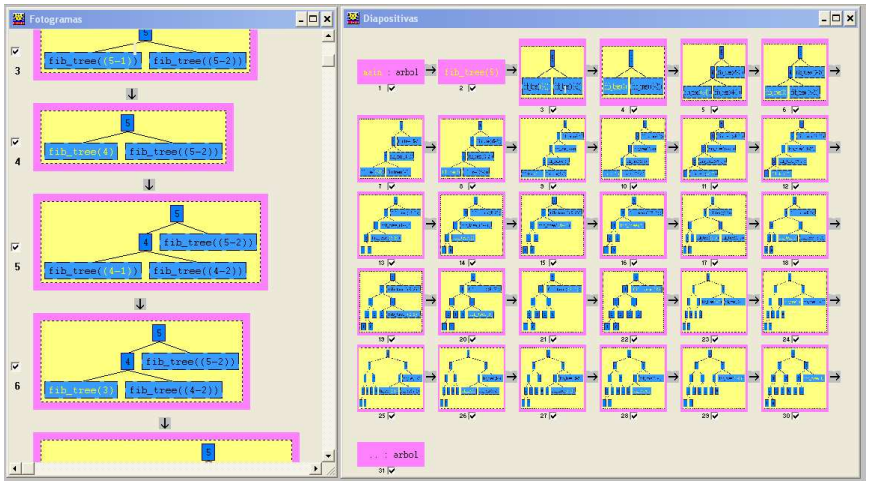



b) Miniaturas nuevas

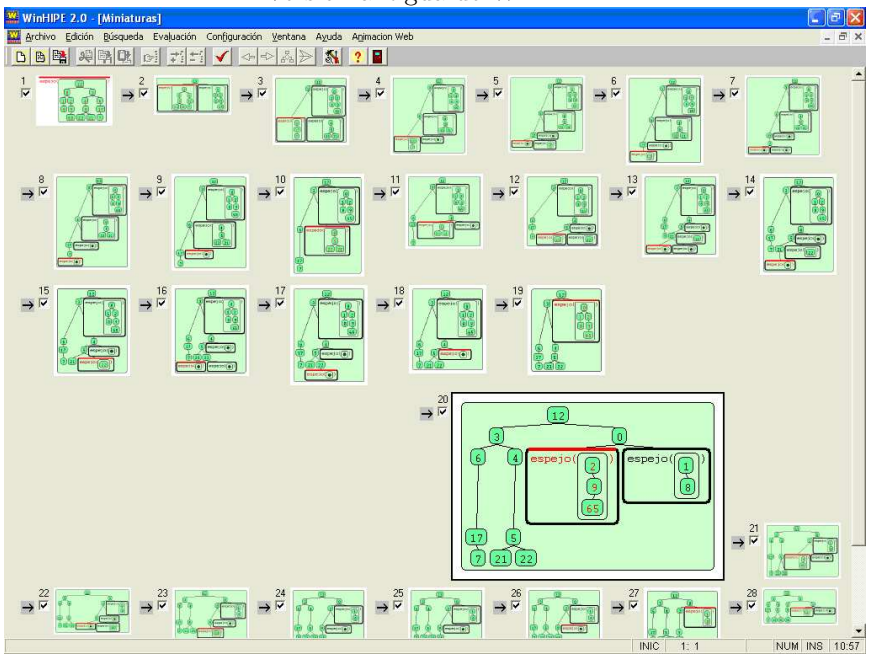
Figura 6.1: Imagen comparativa de la miniaturas antiguas (a), y las nuevas (b). Se puede observar cómo en las miniaturas nuevas, se distingue claramente el flujo de ejecución con un simple vistazo, gracias a las técnicas de pretty-printing. Además el contenido de cada miniatura nueva es más comprensible que su versión antigua

A nivel global, hemos mejorado las posibilidades del usuario para poder ver tantos pasos de ejecución como le sea posible, manteniendo propiedades estructurales intactas, como el orden de ejecución de los pasos. De esta forma, ofrecemos una visión global de la ejecución. Además, permitimos acceder a los detalles de cada paso de ejecución sin perder todo el contexto de la ejecución correspondiente a ese paso. Para ello, hemos estudiado posibles adaptaciones de técnicas de visualización de información existentes, y finalmente hemos desarrollado una nueva técnica llamada *R-Zoom* (véase la figura 6.2).

R-Zoom se fundamenta en dos familias de técnicas de visualización bien conocidas, Zoom+Pan y Focus+Context. Para asegurarnos de cubrir todas las posibilidades hemos evaluado R-Zoom junto con otra alternativa, diseñada según los principios de las familias Zoom+Pan y Overview+Detail. Los resultados de la evaluación indican que usando R-Zoom, y teniendo una pequeña experiencia en las tareas a realizar, se producen mejoras significativas tanto en la eficacia como en la eficiencia. Además, según la opinión de los usuarios R-Zoom es claramente mejor que la alternativa diseñada.




Versión antigua de WinHIPE




Versión de WinHIPE con R-Zoom

Figura 6.2: Mejora del acceso a la vista global y detallada de la ejecución

Desde el punto de vista puramente interactivo hemos simplificado el proceso de generación de animaciones. Las partes automáticas de este proceso –generación de las visualizaciones estáticas, configuración de la apariencia de las visualizaciones– no requieren esfuerzo del usuario. En la parte que el usuario debe realizar –selección de las visualizaciones que aparecerán en la animación–, le proporcionamos una interfaz usable[1] que trabaja con miniaturas de alta calidad. Finalmente, hemos evaluado el impacto de estas mejoras en el esfuerzo que los estudiantes deben dedicar a la construcción de animaciones. Los resultados de dicha evaluación demuestran que este proceso permite construir animaciones sin apenas esfuerzo, permitiendo a los estudiantes centrase en las tareas educativas.

## 6.2. Aspectos pedagógicos

Ahora que contamos con un proceso básico de construcción de animaciones que es sencillo, nos centraremos en las características que deben tener las animaciones para ser herramientas pedagógicas eficaces y sus posibles usos educativos. Nótese que mientras mantengamos la simplicidad en el proceso de construcción, podremos mantener su utilización como ejercicio pedagógico.

### 6.2.1. Características de las animaciones como herramientas pedagógicas

Uno de los problemas de las animaciones generadas con la versión antigua de WinHIPE es su fuerte dependencia de la plataforma. Las animaciones sólo se podían reproducir dentro del entorno. Naharro-Berrocal et al. [147] intentaron solucionar este problema desarrollando las primeras animaciones reproducibles en la Web como un formato alternativo, pero nunca llegaron a integrarlas completamente en el entorno. Nosotros hemos continuado esta línea de trabajo, pero hemos dado a las animaciones reproducibles en la Web[2] un papel protagonista. A partir de ahora, la mayoría de las animaciones serán animaciones Web, que además tendrán su versión reproducible dentro del entorno. Al estar implementadas como documentos HTML con Applets de Java, podrán reproducirse en prácticamente cualquier navegador.

El hecho de utilizar la tecnología HTML facilita, a la vez que condiciona, los aspectos restantes de las animaciones Web. Así, la tecnología HTML facilita la integración de las animaciones con contenidos textuales. Nuestras animaciones Web estarán acompañadas del código fuente del programa, y dos descripciones textuales. Como pensamos en un enfoque educativo, estas descripciones estarán comúnmente asociadas con los objetivos del programa o el problema que resuelva, y con la explicación de la solución propuesta.

Como uno de los usos de las animaciones Web será como material de consulta, y estas son documentos HTML, hemos diseñado su estructura teniendo en cuenta principios de usabilidad de páginas Web. Pero también hemos pensado en ofrecer dos diseños más que, aunque son menos usables, uno es mucho más simple –una página Web plana– y otro más flexible –permitiendo al usuario distribuir toda la información de la animación a su gusto–, cubriendo así tres criterios distintos: simplicidad, usabilidad, y flexibilidad.

La tecnología HTML, permite relacionar unos contenidos con otros mediante hiperenlaces. Nosotros aprovecharemos esta característica para integrar en las propias animaciones toda la información necesaria para manipularlas en el entorno WinHIPE. Ahora, las animaciones son totalmente independientes

[1] eficaz, eficiente y satisfactorio para el usuario
[2] animaciones Web

del usuario que las creó, facilitando su reutilización y mantenimiento.

Desde el punto de vista del profesor, la mejora en los contenidos y calidad de las animaciones motivaría su uso como herramienta educativa de consulta. Por ello, y dado que ningún profesor se quedaría en la producción de una sola animación, proponemos un modelo de gestión de colecciones de animaciones Web. Este modelo facilita la manipulación de estas animaciones utilizando toda la información disponible en ellas. Cabe destacar la facilidad de publicación, así como de modificación de la estructura de las colecciones.

Todas estas mejoras nos han obligado a ampliar el proceso de construcción de las animaciones, ahora animaciones Web. Sin embargo, queremos mantener la simplicidad en el proceso de construcción. Para ello, hemos automatizado al máximo el proceso en tareas como: recoger información sobre las visualizaciones que se mostrarán, el código fuente del programa, la expresión evaluada, la configuración del entorno, o la producción de las distintas páginas Web integrando texto y animación. Pero también hemos simplificado las partes que necesitan de la intervención del usuario: escritura de las dos descripciones textuales asociadas a la animación, y configuración de la apariencia de las páginas Web.

### 6.2.2. Usos educativos de las animaciones

Hemos identificado dos posibles usos educativos de las animaciones Web. Las propias animaciones Web sirven como material de consulta. Su construcción no le llevará mucho trabajo al profesor, y además hemos desarrollado una aplicación capaz de gestionar colecciones de animaciones Web generadas con WinHIPE. Por otro lado, el hecho de construir animaciones es en sí un ejercicio que puede ayudar a aprender. Además, hemos puesto especial atención a la simplicidad del proceso de construcción, por lo tanto, cuando los estudiantes construyan las animaciones se podrán centrar en diseñar la animación más representativa para el programa.

Hemos evaluado ambos usos en dos ámbitos distintos. Hemos hecho una evaluación a corto plazo con un reducido grupo de alumnos en una asignatura de diseño y análisis de algoritmos. Los resultados de esta evaluación confirman que el proceso de construcción de animaciones Web es fácil de aprender y las animaciones son fáciles de construir. Además, aunque los resultados pedagógicos de esta evaluación son difícilmente generalizables, confirman las ideas de Stasko [213] y Hundhausen et al. [94] sobre la eficacia educativa de la construcción de animaciones.

A continuación, hemos realizado una evaluación a largo plazo, durante un curso de programación funcional. Este caso hemos utilizado tres grupos de alumnos. Cada grupo recibió un tratamiento distinto: uno no usó animaciones, otro las usó como material de consulta y finalmente otro construyó animaciones como un ejercicio más de clase. Tras esta evaluación tenemos evidencia significativa de que las animaciones Web son fáciles de usar y construir. Los resultados educativos muestran que los estudiantes que usaron las animaciones, ya sea consultándolas o construyéndolas, obtuvieron mejores calificaciones que los que no las usaron; existiendo leves mejoras en el número de aprobados de los que construyeron frente a los que consultaron. Sin embargo, los estudiantes que construyeron animaciones estaban significativamente más motivados con la asignatura. Además, hemos demostrado que estos resultados son independientes de los profesores que impartieron las clases, y de las titulaciones y turnos en que se encuentran matriculados los estudiantes. Esto, unido al tiempo dedicado a la evaluación, el alto número de estudiantes y el aislamiento de la variable independiente –el nivel de implicación con las animaciones–, da una representatividad alta a los resultados obtenidos.

Lo sorprendente de estos resultados es que prácticamente se aprende lo mismo consultando ani-

maciones que construyéndolas. Según la taxonomía de niveles de implicación de Naps et al. [158], una mayor implicación de los estudiantes con las animaciones resultaría en una mejora del aprendizaje. Construir animaciones supone una mayor implicación que verlas (visión es el nivel de la taxonomía equivalente al uso educativo de consulta que hemos evaluado). Por otro lado, la conclusión más importante de Hundhausen et al. [94] es que el uso más eficaz de las animaciones se corresponde con los enfoques constructivistas, y la construcción de animaciones es precisamente un uso constructivista. De hecho, el aprendizaje mejora si comparamos la construcción con el enfoque típico que no usa las animaciones de ninguna forma.

Entonces, ¿por qué no se aprende más construyendo animaciones que consultándolas? Porque si contemplamos los dos usos educativos al completo, no podemos encontrar una escala que los ordene claramente. La consulta de animaciones tiene un añadido muy importante, las explicaciones textuales. Además, no podemos olvidar que estas animaciones –tanto el componente gráfico como el textual– fueron diseñadas por expertos en la materia que enseñan. Al igual que en la escala del constructivismo la construcción es superior a la consulta; en una escala que considerara la existencia de material adicional como las explicaciones textuales, la consulta es superior a la construcción. Esta última escala se podría asimilar a la teoría cognitiva *Codificación Dual* de Paivio [168]. Esta teoría identifica en las actividades cognitivas dos sistemas simbólicos independientes pero relacionados, el verbal (palabras) y el no verbal (imágenes). Según Mayer y Anderson [134], el uso de animaciones gráficas junto con explicaciones textuales mejoraría la adquisición del conocimiento con respecto a animaciones que sólo tuvieran representaciones gráficas.

Por lo tanto, los dos usos educativos de las animaciones –consulta y construcción– no se pueden clasificar totalmente en una escala donde la consulta es menos que la construcción. Sencillamente son diferentes enfoques pedagógicos, y sus resultados no tienen porqué seguir un orden prefijado.

De hecho, nuestros resultados nos llevan a pensar que ambos enfoques se complementan entre sí, estableciendo un modelo de utilización de animaciones mixto. En escenarios más "expositivos" –clases magistrales, publicación de material para su uso por parte de los alumnos– la consulta parece una forma adecuada, y según nuestros resultados más efectiva que el enfoque típico. La construcción de animaciones se adapta mejor a escenarios más "activos" –laboratorios de ejercicios, prácticas individuales o en grupo–, como también lo han demostrado nuestros resultados con respecto al enfoque típico.

Por lo tanto, podemos concluir que ambos usos educativos, consulta y construcción, son en conjunto una herramienta pedagógica de bastante potencia. Hemos comprobado que mejora el aprendizaje de los estudiantes, aumenta su motivación hacia la asignatura, y es reconocida por estos como una herramienta que requiere poco esfuerzo para su uso y que les ayuda en el aprendizaje.

## 6.3. Trabajos futuros

Las líneas de trabajo futuro que hemos identificado están relacionadas con los modelos expuestos en el marco de trabajo, junto con el uso educativo de las animaciones. Como es lógico, la parte fundamental se encuentra en la construcción y el uso educativo de las animaciones. Sin embargo, hemos detectado algunos puntos mejorables en el resto de modelos.

### 6.3.1. Modelos de ejecución, visualización y reproducción de animaciones

Las facilidades disponibles en el modelo de ejecución se centran en el siguiente paso a ejecutar. Desde un punto de vista más global, sería recomendable poder trazar la evolución de una subexpresión de la expresión inicial, pudiendo dar respuesta a preguntas típicas como ¿de dónde viene este valor?

El modelo de visualización mezcla el código fuente y los datos gracias a la visualización de las expresiones, característica del paradigma funcional. Sin embargo, se podría hacer más explícita la relación entre el código fuente y su ejecución usando modelos de otros paradigmas. Por ejemplo, se podría resaltar la parte correspondiente del programa fuente que se está evaluando en la expresión, o se podría mostrar el árbol de activación.

Según el modelo de reproducción de animaciones actual, la animación se consigue mostrando secuencialmente las visualizaciones estáticas con transiciones discretas. Se podrían generar transiciones suaves entre las visualizaciones consecutivas; así se aprovecharía el esfuerzo invertido en comprender la visualización anterior para la siguiente.

### 6.3.2. Modelo de construcción de las animaciones

Según nuestros resultados, la construcción de animaciones de programas ayuda al aprendizaje. Una de las claves de este hecho se encuentra en la simplicidad del proceso de construcción. Identificamos dos líneas de trabajo relacionadas con el proceso de construcción: su utilización fuera de la enseñanza de la programación, y su mejora mediante cambios internos al proceso.

Una de las aportaciones fundamentales de esta tesis es la técnica de visualización de información R-Zoom. Esta ha demostrado ser una técnica eficaz, eficiente y satisfactoria para el usuario, en el entorno de construcción de animaciones programas funcionales. Sin embargo, creemos que su ámbito de aplicación puede ser más extenso, tanto a nivel educativo como más general. Actualmente estamos trabajando en la generalización de la técnica, de forma que su uso en otro ámbito no requiera de nuevo su implementación. El lenguaje de programación utilizado es Java, dado la extensión de su uso y sus capacidades para el desarrollo de API's. La idea principal es encapsular en un *Layout* las características de distribución de los elementos en el espacio de trabajo, y en una *Interfaz* las facilidades que deberían ofrecer los elementos a utilizar con R-Zoom, p.ej., versiones de contexto y detalle. A nivel educativo, cualquier entorno capaz de generar animaciones podría hacer uso de este API, facilitando así la construcción. A nivel más general, siempre que se den las condiciones de diseño de R-Zoom –espacios de trabajo con gran número de elementos a manipular, organizados de forma ordenada, y con necesidades de visiones global y detallada de los mismos–, se puede pensar en su utilización.

Por otro lado, existen algunos puntos del proceso de construcción cuya mejora puede dar hacer aún más simple la generación de animaciones.

- En R-Zoom, los cambios de foco son transiciones discretas e inmediatas, y su suavización [44] podría mejorar la percepción del contexto. Un ejemplo sería animar los movimientos de traslación que sufren los elementos del contexto, así como los cambios de tamaño del foco.

- Cuando una miniatura es demasiado grande, se permite resumirla utilizando técnicas de ojo de pez [69, 70]; de esta forma se elimina el contenido que no cumpla ciertos criterios. Una alternativa posible sería utilizar la técnica de ojo de pez gráfico [198]. Esta técnica distorsionaría los contenidos, de forma que el redex ocuparía mayor espacio por ser la parte más importante, y el resto de la visualización se adaptaría al espacio restante.

- En R-Zoom representamos el contexto con las miniaturas, versiones reducidas en escala de las visualizaciones originales. Otra posible representación del contexto sería generar visualizaciones resumen de un conjunto de ellas. Como se ha explicado anteriormente, la ejecución de los programas funcionales se basa en un proceso de modificación de expresiones; por lo tanto, una expresión y la siguiente podrían tener parte común; y su resumen podría ser la parte común junto con la parte distinta resaltada de algún modo. Esto nos llevaría a la creación de una jerarquía de visualizaciones resumen; cuanto más alto en la jerarquía, mayores son las partes distintas; cuanto más cercano a las hojas, mayores serían las partes comunes.

### 6.3.3. Uso educativo de las animaciones

En todas las evaluaciones que hemos hecho, los sujetos de estudio eran estudiantes. Sin embargo, las animaciones generadas para su consulta, fueron generadas por profesores. Aunque la simplicidad del proceso de construcción es independiente del usuario en sí, no hemos realizado ninguna evaluación de la usabilidad de esta herramienta desde el punto de vista de los profesores. Esta línea de trabajo es importante, puesto que son los profesores quienes toman la decisión de incluir las animaciones en su metodología docente, y su adopción depende en gran manera del esfuerzo [96, 107, 155] que tengan que dedicarle a esta tarea. Nos proponemos realizar evaluaciones del esfuerzo dedicado a la generación y mantenimiento de animaciones, así como colecciones de estas.

La integración de exlicaciones textuales ha sido un factor clave en la eficacia pedagógica de las animaciones presentadas en esta tesis. Siguiendo esta idea, se está trabajando en la integración de las animaciones con distintos materiales, como libros electrónicos [159, 194].

En esta tesis nos hemos centrado tanto de la consulta de animaciones, como en su construcción partiendo de programas ya codificados. Sin embargo, hacer el programa que resuelve el problema propuesto es el fin de las asignaturas de programación. La continuación natural de este trabajo sería investigar el impacto de añadir la generación de animaciones a la tarea de codificar el problema que resuelve un problema propuesto. Este nuevo uso de las animaciones obligaría a los estudiantes no sólo a codificar el programa, sino a generar una animación que demuestre y explique su funcionamiento. Así conseguiríamos que la implicación de los estudiantes fuera todavía más activa, y en términos constructivistas, esto debería mejorar el aprendizaje.

Fruto de nuestro trabajo con las visualizaciones, hemos desarrollado una guía para la construcción de animaciones Web de algoritmos [237]. El siguiente paso consisterá en la evaluación de dicha guía desde el punto de vista del profesor.

Finalmente, un campo de interés creciente es la adaptación de las animaciones a los estudiantes que las utilizan [38, 130]. Realmente, nace de la evolución natural de los materiales educativos dinámicos, hacia materiales adaptativos. La idea principal es estructurar las animaciones identificando conceptos utilizados en ellas, incluso a nivel de fotograma o paso [159, 160]. Así se puede adaptar lo que se muestra a lo que el estudiante sabe. Por otro lado está la forma de obtener la información sobre el conocimiento de estudiante, que puede ser simplemente si ya ha visto pasos similares anteriormente en la misma animación [38], o según las respuestas a preguntas integradas en la animación [38, 78], u obteniendo información sobre los usuarios a partir de bases de datos.

# Apéndice A

# Detalles de la evaluación de R-Zoom

## A.1. Cuestionario de opinión sobre la evaluación de *R-Zoom*

Para cada pregunta, selecciona la opción que te parezca más oportuna, redondeando la letra correspondiente.

1. ¿Crees que una utilidad de zoom ayuda a la generación de animaciones con WinHIPE?

   **a)** Sí.

   **b)** Algo, pero no mucho.

   **c)** Sin opinión.

   **d)** No.

2. De los zoom con los que has trabajado ¿cuál te gusta más?:

   **a)** El primero.

   **b)** El segundo.

   **c)** Los dos por igual.

   **d)** Ninguno.

3. Opina sobre la utilidad de los dos zoom que has probado:

   **a)** El primero es mucho más útil que el segundo.

   **b)** El primero es un poco más útil que el segundo.

   **c)** Son iguales.

   **d)** El segundo es un poco más útil que el primero.

   **e)** El segundo es mucho más útil que el primero.

4. Opina sobre la facilidad de uso de los dos zoom que has probado:

   **a)** El primero es mucho más fácil de usar que el segundo.

   **b)** El primero es un poco más fácil de usar que el segundo.

   **c)** Son iguales.

   **d)** El segundo es un poco más fácil de usar que el primero.

   **e)** El segundo es mucho más fácil de usar que el primero.

5. ¿Qué características te parecen más posistivas de los zooms con los que has trabajado?

6. ¿Qué características te parecen más negativas de los zooms con los que has trabajado?

## A.2. Análisis de validez de los datos

En este apéndice detallamos el análisis estadístico realizado para comprobar que los datos recogidos de cada turno, en cuanto a tiempo invertido y errores cometidos, pertenecen a la misma población pudiendo ser tratados como un mismo grupo.

En las figuras A.1 y A.2 mostramos el análisis de los datos de errores cometidos por los usuarios del grupo A y B respectivamente. Dichos datos no siguen una distribución normal, por lo que hemos utilizado el test de *Mann-Whitney*. Todos los valores de $p$, última fila de las tablas, son mayores de `0,05`, por lo tanto cada pareja de turnos asociados a cada grupo puede tratarse como un solo grupo, en cuanto a errores cometidos se refiere.

**Test Statistics[b]**

| | ENTREN. | ARZ11 | ARZ12 | ARZ13 | ARZ21 | ARZ22 |
|---|---|---|---|---|---|---|
| Mann-Whitney U | 49,000 | 54,500 | 52,500 | 57,000 | 58,000 | 55,500 |
| Wilcoxon W | 77,000 | 244,500 | 80,500 | 247,000 | 248,000 | 245,500 |
| Z | -1,472 | -1,109 | -1,294 | -1,648 | -,719 | -1,014 |
| Asymp. Sig. (2-tailed) | ,141 | ,267 | ,196 | ,099 | ,472 | ,311 |
| Exact Sig. [2*(1-tailed Sig.)] | ,334[a] | ,497[a] | ,427[a] | ,611[a] | ,651[a] | ,534[a] |

**Test Statistics[b]**

| | ARZ23 | ARZ31 | ARZ32 | ARZ33 | AOD11 | AOD12 |
|---|---|---|---|---|---|---|
| Mann-Whitney U | 59,500 | 63,000 | 66,500 | 59,500 | 66,500 | 60,500 |
| Wilcoxon W | 87,500 | 91,000 | 94,500 | 87,500 | 94,500 | 250,500 |
| Z | -,876 | -,607 | ,000 | -,876 | ,000 | -,751 |
| Asymp. Sig. (2-tailed) | ,381 | ,544 | 1,000 | ,381 | 1,000 | ,453 |
| Exact Sig. [2*(1-tailed Sig.)] | ,692[a] | ,866[a] | 1,000[a] | ,692[a] | 1,000[a] | ,735[a] |

**Test Statistics[b]**

| | AOD13 | AOD21 | AOD22 | AOD23 | AOD31 |
|---|---|---|---|---|---|
| Mann-Whitney U | 66,500 | 59,500 | 48,500 | 61,000 | 60,500 |
| Wilcoxon W | 94,500 | 87,500 | 238,500 | 251,000 | 250,500 |
| Z | ,000 | -,876 | -1,425 | -,688 | -,751 |
| Asymp. Sig. (2-tailed) | 1,000 | ,381 | ,154 | ,492 | ,453 |
| Exact Sig. [2*(1-tailed Sig.)] | 1,000[a] | ,692[a] | ,306[a] | ,778[a] | ,735[a] |

**Test Statistics[b]**

| | AOD32 | AOD33 |
|---|---|---|
| Mann-Whitney U | 56,000 | 56,000 |
| Wilcoxon W | 84,000 | 84,000 |
| Z | -1,096 | -1,094 |
| Asymp. Sig. (2-tailed) | ,273 | ,274 |
| Exact Sig. [2*(1-tailed Sig.)] | ,572[a] | ,572[a] |

a. Not corrected for ties.

Figura A.1: Grupo A, datos de errores

**Test Statistics[b]**

| | ENTREN. | BOD11 | BOD12 | BOD13 | BOD21 | BOD22 |
|---|---|---|---|---|---|---|
| Mann-Whitney U | 36,000 | 31,500 | 28,000 | 31,500 | 36,000 | 31,500 |
| Wilcoxon W | 81,000 | 76,500 | 64,000 | 76,500 | 81,000 | 76,500 |
| Z | ,000 | -1,061 | -1,377 | -1,061 | ,000 | -1,061 |
| Asymp. Sig. (2-tailed) | 1,000 | ,289 | ,168 | ,289 | 1,000 | ,289 |
| Exact Sig. [2*(1-tailed Sig.)] | 1,000[a] | ,673[a] | ,481[a] | ,673[a] | 1,000[a] | ,673[a] |

**Test Statistics[b]**

| | BOD23 | BOD31 | BOD32 | BOD33 | BRZ11 | BRZ12 |
|---|---|---|---|---|---|---|
| Mann-Whitney U | 32,000 | 28,000 | 28,000 | 35,500 | 36,000 | 31,500 |
| Wilcoxon W | 68,000 | 64,000 | 64,000 | 80,500 | 81,000 | 76,500 |
| Z | -,943 | -1,377 | -1,374 | -,086 | ,000 | -1,061 |
| Asymp. Sig. (2-tailed) | ,346 | ,168 | ,169 | ,931 | 1,000 | ,289 |
| Exact Sig. [2*(1-tailed Sig.)] | ,743[a] | ,481[a] | ,481[a] | ,963[a] | 1,000[a] | ,673[a] |

**Test Statistics[b]**

| | BRZ13 | BRZ21 | BRZ22 | BRZ23 | BRZ31 |
|---|---|---|---|---|---|
| Mann-Whitney U | 31,500 | 36,000 | 36,000 | 36,000 | 32,000 |
| Wilcoxon W | 76,500 | 81,000 | 81,000 | 81,000 | 68,000 |
| Z | -1,061 | ,000 | ,000 | ,000 | -,943 |
| Asymp. Sig. (2-tailed) | ,289 | 1,000 | 1,000 | 1,000 | ,346 |
| Exact Sig. [2*(1-tailed Sig.)] | ,673[a] | 1,000[a] | 1,000[a] | 1,000[a] | ,743[a] |

**Test Statistics[b]**

| | BRZ32 | BRZ33 |
|---|---|---|
| Mann-Whitney U | 31,500 | 36,000 |
| Wilcoxon W | 76,500 | 81,000 |
| Z | -1,061 | ,000 |
| Asymp. Sig. (2-tailed) | ,289 | 1,000 |
| Exact Sig. [2*(1-tailed Sig.)] | ,673[a] | 1,000[a] |

a. Not corrected for ties.

Figura A.2: Grupo B, datos de errores

En las figura A.3 y A.4 mostramos el análisis de los datos del tiempo empleado en realizar las tareas, por los usuarios del grupo A y B respectivamente. Dichos datos siguen una distribución normal, por lo que hemos utilizado el test *t-student para igualdad de medias*. En el caso del grupo A, hemos detectado una tarea con diferencias significativas, en el grupo B tres. En la tabla A.1 mostramos los detalles estadísticos de dichas tareas.

| | | Levene's Test for Equality of Variances | | t-test for Equality of Means | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | 95% Confidence Interval of the Difference | |
| | | F | Sig. | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference | Lower | Upper |
| ENTREN. | Equal variances assumed | 5,332 | ,030 | 1,665 | 24 | ,109 | 59,21805 | 35,57208 | -14,19912 | 132,63521 |
| | Equal variances not assumed | | | 2,444 | 23,993 | ,022 | 59,21805 | 24,22506 | 9,21915 | 109,21694 |
| ARZ11 | Equal variances assumed | 6,343 | ,019 | 1,347 | 24 | ,191 | 49,23308 | 36,56366 | -26,23061 | 124,69678 |
| | Equal variances not assumed | | | 2,042 | 23,673 | ,052 | 49,23308 | 24,11321 | -,57060 | 99,03677 |
| ARZ12 | Equal variances assumed | ,458 | ,505 | ,837 | 24 | ,411 | 10,36090 | 12,37366 | -15,17707 | 35,89887 |
| | Equal variances not assumed | | | 1,107 | 20,622 | ,281 | 10,36090 | 9,36282 | -9,13191 | 29,85371 |
| ARZ13 | Equal variances assumed | 2,128 | ,158 | -,968 | 24 | ,343 | -5,66165 | 5,84645 | -17,72814 | 6,40483 |
| | Equal variances not assumed | | | -,789 | 7,898 | ,453 | -5,66165 | 7,17504 | -22,24470 | 10,92139 |
| ARZ21 | Equal variances assumed | 11,522 | ,002 | -1,525 | 24 | ,140 | -25,81203 | 16,93131 | -60,75654 | 9,13248 |
| | Equal variances not assumed | | | -1,071 | 6,810 | ,321 | -25,81203 | 24,09629 | -83,11433 | 31,49027 |
| ARZ22 | Equal variances assumed | 1,132 | ,298 | 1,265 | 24 | ,218 | 17,81203 | 14,07699 | -11,24146 | 46,86552 |
| | Equal variances not assumed | | | 1,769 | 23,006 | ,090 | 17,81203 | 10,06803 | -3,01496 | 38,63902 |
| ARZ23 | Equal variances assumed | ,606 | ,444 | 1,564 | 24 | ,131 | 16,90226 | 10,81004 | -5,40858 | 39,21309 |
| | Equal variances not assumed | | | 1,793 | 14,458 | ,094 | 16,90226 | 9,42612 | -3,25486 | 37,05937 |
| ARZ31 | Equal variances assumed | 1,717 | ,202 | ,559 | 24 | ,581 | 24,32331 | 43,50500 | -65,46660 | 114,11322 |
| | Equal variances not assumed | | | ,434 | 7,479 | ,677 | 24,32331 | 56,06019 | -106,53474 | 155,18136 |
| ARZ32 | Equal variances assumed | ,059 | ,810 | 1,596 | 24 | ,124 | 22,10526 | 13,85114 | -6,48209 | 50,69262 |
| | Equal variances not assumed | | | 1,489 | 9,521 | ,169 | 22,10526 | 14,85064 | -11,21125 | 55,42178 |
| ARZ33 | Equal variances assumed | 7,747 | ,010 | ,289 | 24 | ,775 | 6,59398 | 22,83609 | -40,53738 | 53,72535 |
| | Equal variances not assumed | | | ,386 | 21,198 | ,703 | 6,59398 | 17,06786 | -28,88038 | 42,06835 |
| AOD11 | Equal variances assumed | ,411 | ,528 | -1,008 | 24 | ,323 | -81,26316 | 80,61569 | -247,64577 | 85,11945 |
| | Equal variances not assumed | | | -1,230 | 16,889 | ,236 | -81,26316 | 66,08280 | -220,75571 | 58,22940 |
| AOD12 | Equal variances assumed | ,365 | ,551 | -2,261 | 24 | ,033 | -21,34586 | 9,44235 | -40,83391 | -1,85782 |
| | Equal variances not assumed | | | -2,245 | 10,612 | ,047 | -21,34586 | 9,50734 | -42,36499 | -,32674 |
| AOD13 | Equal variances assumed | ,472 | ,499 | -1,329 | 24 | ,196 | -20,29323 | 15,27043 | -51,80985 | 11,22338 |
| | Equal variances not assumed | | | -1,257 | 9,745 | ,238 | -20,29323 | 16,14041 | -56,38409 | 15,79762 |
| AOD21 | Equal variances assumed | ,070 | ,794 | -,808 | 24 | ,427 | -18,14286 | 22,44870 | -64,47471 | 28,18899 |
| | Equal variances not assumed | | | -,903 | 13,590 | ,382 | -18,14286 | 20,08883 | -61,35133 | 25,06562 |
| AOD22 | Equal variances assumed | 1,550 | ,225 | -,430 | 24 | ,671 | -5,74436 | 13,36390 | -33,32609 | 21,83737 |
| | Equal variances not assumed | | | -,528 | 17,151 | ,605 | -5,74436 | 10,88917 | -28,70308 | 17,21436 |
| AOD23 | Equal variances assumed | ,237 | ,630 | -,159 | 24 | ,875 | -1,62406 | 10,21107 | -22,69868 | 19,45056 |
| | Equal variances not assumed | | | -,153 | 9,987 | ,882 | -1,62406 | 10,63738 | -25,32976 | 22,08164 |
| AOD31 | Equal variances assumed | ,132 | ,719 | -1,180 | 24 | ,250 | -24,09774 | 20,43009 | -66,26337 | 18,06788 |
| | Equal variances not assumed | | | -1,282 | 12,745 | ,223 | -24,09774 | 18,80386 | -64,80389 | 16,60840 |
| AOD32 | Equal variances assumed | 6,259 | ,020 | -1,681 | 24 | ,106 | -39,03759 | 23,22445 | -86,97050 | 8,89531 |
| | Equal variances not assumed | | | -1,254 | 7,185 | ,249 | -39,03759 | 31,14143 | -112,29187 | 34,21668 |
| AOD33 | Equal variances assumed | ,116 | ,737 | ,564 | 24 | ,578 | 11,41353 | 20,23937 | -30,35848 | 53,18555 |
| | Equal variances not assumed | | | ,550 | 10,266 | ,594 | 11,41353 | 20,75454 | -34,66846 | 57,49553 |

Figura A.3: Grupo A, datos de tiempo empleado en realizar las tareas

| | | Levene's Test for Equality of Variances | | t-test for Equality of Means | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | 95% Confidence Interval of the Difference | |
| | | F | Sig. | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference | Lower | Upper |
| ENTREN. | Equal variances assumed | 9,525 | ,008 | -1,506 | 15 | ,153 | -34,30556 | 22,78017 | -82,86035 | 14,24924 |
| | Equal variances not assumed | | | -1,586 | 9,632 | ,145 | -34,30556 | 21,62583 | -82,74183 | 14,13072 |
| BOD11 | Equal variances assumed | 6,193 | ,025 | ,684 | 15 | ,504 | 23,86111 | 34,87810 | -50,47981 | 98,20203 |
| | Equal variances not assumed | | | ,708 | 12,552 | ,492 | 23,86111 | 33,69764 | -49,20318 | 96,92540 |
| BOD12 | Equal variances assumed | ,006 | ,940 | ,972 | 15 | ,347 | 16,45833 | 16,93733 | -19,64272 | 52,55939 |
| | Equal variances not assumed | | | ,959 | 13,444 | ,355 | 16,45833 | 17,17024 | -20,51163 | 53,42830 |
| BOD13 | Equal variances assumed | ,607 | ,448 | ,406 | 15 | ,690 | 3,65278 | 8,99348 | -15,51637 | 22,82193 |
| | Equal variances not assumed | | | ,413 | 14,734 | ,686 | 3,65278 | 8,85003 | -15,24030 | 22,54586 |
| BOD21 | Equal variances assumed | 1,073 | ,317 | ,021 | 15 | ,983 | ,52778 | 24,77775 | -52,28475 | 53,34031 |
| | Equal variances not assumed | | | ,021 | 13,304 | ,984 | ,52778 | 25,14578 | -53,67043 | 54,72599 |
| BOD22 | Equal variances assumed | 3,023 | ,103 | 2,722 | 15 | ,016 | 20,54167 | 7,54743 | 4,45469 | 36,62864 |
| | Equal variances not assumed | | | 2,675 | 12,969 | ,019 | 20,54167 | 7,67880 | 3,94855 | 37,13478 |
| BOD23 | Equal variances assumed | ,613 | ,446 | ,882 | 15 | ,391 | 9,84722 | 11,15938 | -13,93843 | 33,63288 |
| | Equal variances not assumed | | | ,893 | 14,945 | ,386 | 9,84722 | 11,03096 | -13,67224 | 33,36668 |
| BOD31 | Equal variances assumed | ,002 | ,968 | ,672 | 15 | ,512 | 12,65278 | 18,82368 | -27,46894 | 52,77450 |
| | Equal variances not assumed | | | ,666 | 13,970 | ,516 | 12,65278 | 18,99847 | -28,10314 | 53,40869 |
| BOD32 | Equal variances assumed | ,074 | ,789 | ,247 | 15 | ,808 | 5,87500 | 23,79645 | -44,84594 | 56,59594 |
| | Equal variances not assumed | | | ,246 | 14,379 | ,809 | 5,87500 | 23,92008 | -45,30192 | 57,05192 |
| BOD33 | Equal variances assumed | ,893 | ,360 | 1,404 | 15 | ,181 | 25,27778 | 18,00963 | -13,10884 | 63,66439 |
| | Equal variances not assumed | | | 1,376 | 12,632 | ,193 | 25,27778 | 18,36766 | -14,52113 | 65,07668 |
| BRZ11 | Equal variances assumed | 1,247 | ,282 | 2,486 | 15 | ,025 | 183,80556 | 73,93282 | 26,22147 | 341,38964 |
| | Equal variances not assumed | | | 2,457 | 13,681 | ,028 | 183,80556 | 74,80595 | 23,01127 | 344,59984 |
| BRZ12 | Equal variances assumed | 3,401 | ,085 | 2,409 | 15 | ,029 | 19,34722 | 8,03196 | 2,22751 | 36,46693 |
| | Equal variances not assumed | | | 2,341 | 11,316 | ,039 | 19,34722 | 8,26625 | 1,21520 | 37,47925 |
| BRZ13 | Equal variances assumed | ,312 | ,585 | ,932 | 15 | ,366 | 5,88889 | 6,31673 | -7,57489 | 19,35267 |
| | Equal variances not assumed | | | ,934 | 14,875 | ,365 | 5,88889 | 6,30322 | -7,55594 | 19,33372 |
| BRZ21 | Equal variances assumed | ,051 | ,825 | ,336 | 15 | ,742 | 4,48611 | 13,35966 | -23,98933 | 32,96155 |
| | Equal variances not assumed | | | ,335 | 14,624 | ,742 | 4,48611 | 13,38881 | -24,11552 | 33,08774 |
| BRZ22 | Equal variances assumed | ,850 | ,371 | ,388 | 15 | ,704 | 2,98611 | 7,70113 | -13,42846 | 19,40068 |
| | Equal variances not assumed | | | ,393 | 14,818 | ,700 | 2,98611 | 7,58917 | -13,20713 | 19,17935 |
| BRZ23 | Equal variances assumed | 1,641 | ,220 | ,865 | 15 | ,401 | 10,13889 | 11,72543 | -14,85328 | 35,13106 |
| | Equal variances not assumed | | | ,825 | 8,811 | ,431 | 10,13889 | 12,28818 | -17,74984 | 38,02762 |
| BRZ31 | Equal variances assumed | ,255 | ,621 | ,227 | 15 | ,824 | 8,68056 | 38,24416 | -72,83493 | 90,19604 |
| | Equal variances not assumed | | | ,228 | 14,957 | ,823 | 8,68056 | 38,07145 | -72,48698 | 89,84809 |
| BRZ32 | Equal variances assumed | 1,020 | ,329 | ,291 | 15 | ,775 | 3,30556 | 11,37058 | -20,93026 | 27,54137 |
| | Equal variances not assumed | | | ,301 | 12,527 | ,768 | 3,30556 | 10,98403 | -20,51531 | 27,12642 |
| BRZ33 | Equal variances assumed | 2,389 | ,143 | -,643 | 15 | ,530 | -18,06944 | 28,11586 | -77,99698 | 41,85809 |
| | Equal variances not assumed | | | -,660 | 13,781 | ,520 | -18,06944 | 27,39686 | -76,91772 | 40,77883 |

Figura A.4: Grupo B, datos de tiempo empleado en realizar las tareas

| Tarea | Valores $t$ y $p$ | Turno 1 | Turno 2 |
|---|---|---|---|
| AOD12 | $t(24) = -2{,}261, p = 0{,}033$ | $(M = 28{,}368, SD = 21{,}2791)$ | $(M = 49{,}714, SD = 21{,}5848)$ |
| BOD22 | $t(15) = 2{,}722, p = 0{,}016$ | $(M = 53{,}875, SD = 17{,}6994)$ | $(M = 33,333, SD = 13{,}3510)$ |
| BRZ11 | $t(15) = 2{,}486, p = 0{,}025$ | $(M = 454{,}25, SD = 166{,}8102)$ | $(M = 270{,}444, SD = 138{,}0562)$ |
| BRZ12 | $t(15) = 2{,}409, p = 0{,}029$ | $(M = 44{,}125, SD = 20{,}2656)$ | $(M = 24{,}777, SD = 12{,}3671)$ |

Tabla A.1: Detalle del análisis estadístico de igualdad de medias para las tareas con diferencias significativas entre turnos

Para comprobar la posible influencia de estas anomalías en otros análisis hemos verificado la dependencia entre el tiempo y, la colección y la imagen de cada tarea. Si no son independientes, entoces todo análisis comparativo deberá tener en cuenta los datos ignorados, y por lo tanto ignorar los datos homólogos. Utilizaremos los test de correlacción $\tau - b$ de Kendall y $\rho$ de Spearman. En la figura A.5 mostramos el análisis de dependencias para ambos grupos. Los resultados muestran que los tiempos dependen tanto de la colección utilizada como de la imagen objetivo.

**Correlations**

| | | | PHASE | COLECTION | IMAGE | TIME |
|---|---|---|---|---|---|---|
| Kendall's tau_b | TIME | Correlation Coefficient | -,019 | ,282** | -,448** | 1,000 |
| | | Sig. (2-tailed) | ,631 | ,000 | ,000 | . |
| | | N | 416 | 416 | 416 | 416 |
| Spearman's rho | TIME | Correlation Coefficient | -,024 | ,354** | -,578** | 1,000 |
| | | Sig. (2-tailed) | ,631 | ,000 | ,000 | . |
| | | N | 416 | 416 | 416 | 416 |

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

Análisis de correlacción en el grupo A

**Correlations**

| | | | PHASE | COLECTION | IMAGE | TIME |
|---|---|---|---|---|---|---|
| Kendall's tau_b | TIME | Correlation Coefficient | -,230** | ,195** | -,485** | 1,000 |
| | | Sig. (2-tailed) | ,000 | ,000 | ,000 | . |
| | | N | 255 | 255 | 255 | 255 |
| Spearman's rho | TIME | Correlation Coefficient | -,280** | ,247** | -,623** | 1,000 |
| | | Sig. (2-tailed) | ,000 | ,000 | ,000 | . |
| | | N | 255 | 255 | 255 | 255 |

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

Análisis de correlacción en el grupo B

Figura A.5: Análisis de correlacción entre el tiempo invertido en las tareas; y la fase, la colección y la imagen de dichas tareas

## A.3. Análisis de diferencias significativas en eficacia

Aquí detallamos el análisis de diferencias significativas según el análisis por grupo-interfaz (GIB) y por tarea (TB). Los test utilizados son el test de *Mann-Whitney* para comparar resultados entre los grupos A y B, y el test de *Wilcoxon* para comparar resultados dentro del mismo grupo pero con diferentes interfaces.

### A.3.1. Análisis GIB

Como podemos observar en las figuras A.6 y A.7, las tres comparaciones que se hacen con BRZ dan resultados significativos.

**Ranks**

| | Group | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| BODvsAOD | 0 | 153 | 193,16 | 29554,00 |
| | 1 | 234 | 194,55 | 45524,00 |
| | Total | 387 | | |
| BODvsARZ | 0 | 153 | 191,71 | 29332,00 |
| | 1 | 234 | 195,50 | 45746,00 |
| | Total | 387 | | |
| BRZvsAOD | 0 | 153 | 187,03 | 28616,00 |
| | 1 | 234 | 198,56 | 46462,00 |
| | Total | 387 | | |
| BRZvsARZ | 0 | 153 | 185,55 | 28388,50 |
| | 1 | 234 | 199,53 | 46689,50 |
| | Total | 387 | | |

**Test Statistics[a]**

| | BODvsAOD | BODvsARZ | BRZvsAOD | BRZvsARZ |
|---|---|---|---|---|
| Mann-Whitney U | 17773,000 | 17551,000 | 16835,000 | 16607,500 |
| Wilcoxon W | 29554,000 | 29332,000 | 28616,000 | 28388,500 |
| Z | -,249 | -,655 | -2,371 | -2,724 |
| Asymp. Sig. (2-tailed) | ,803 | ,513 | ,018 | ,006 |

a. Grouping Variable: Group

Figura A.6: Test de *Mann-Whitney* para diferencias significativas (GIB) entre datos sobre errores cometidos por usuarios de diferentes grupos

**Ranks**

| | | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| BRZ - BOD | Negative Ranks | 12[a] | 8,67 | 104,00 |
| | Positive Ranks | 4[b] | 8,00 | 32,00 |
| | Ties | 137[c] | | |
| | Total | 153 | | |
| AOD - ARZ | Negative Ranks | 21[d] | 18,50 | 388,50 |
| | Positive Ranks | 17[e] | 20,74 | 352,50 |
| | Ties | 196[f] | | |
| | Total | 234 | | |

a. BRZ < BOD
b. BRZ > BOD
c. BRZ = BOD
d. AOD < ARZ
e. AOD > ARZ
f. AOD = ARZ

**Test Statistics[b]**

| | BRZ - BOD | AOD - ARZ |
|---|---|---|
| Z | -2,065[a] | -,293[a] |
| Asymp. Sig. (2-tailed) | ,039 | ,770 |

a. Based on positive ranks.
b. Wilcoxon Signed Ranks Test

Figura A.7: Test de *Wilcoxon* para diferencias significativas (GIB) entre datos sobre errores cometidos por usuarios del mismo grupo usando diferentes interfaces

### A.3.2. Análisis TB

**Test Statistics[a]**

| | BOD-ARZ-11 | BOD-ARZ-12 | BOD-ARZ-13 | BOD-ARZ-21 | BOD-ARZ-22 |
|---|---|---|---|---|---|
| Mann-Whitney U | 200,000 | 213,000 | 216,500 | 178,500 | 199,500 |
| Wilcoxon W | 353,000 | 366,000 | 567,500 | 331,500 | 352,500 |
| Z | -,939 | -,331 | -,306 | -1,901 | -,960 |
| Asymp. Sig. (2-tailed) | ,348 | ,741 | ,759 | ,057 | ,337 |

**Test Statistics[a]**

| | BOD-ARZ-23 | BOD-ARZ-31 | BOD-ARZ-32 | BOD-ARZ-33 | BOD-AOD-11 |
|---|---|---|---|---|---|
| Mann-Whitney U | 217,000 | 203,500 | 195,000 | 212,000 | 208,000 |
| Wilcoxon W | 370,000 | 554,500 | 546,000 | 563,000 | 559,000 |
| Z | -,225 | -,985 | -1,770 | -,444 | -1,237 |
| Asymp. Sig. (2-tailed) | ,822 | ,325 | ,077 | ,657 | ,216 |

**Test Statistics[a]**

| | BOD-AOD-12 | BOD-AOD-13 | BOD-AOD-21 | BOD-AOD-22 | BOD-AOD-23 |
|---|---|---|---|---|---|
| Mann-Whitney U | 212,000 | 208,000 | 204,000 | 183,000 | 216,500 |
| Wilcoxon W | 563,000 | 559,000 | 357,000 | 336,000 | 369,500 |
| Z | -,444 | -1,237 | -1,157 | -1,476 | -,253 |
| Asymp. Sig. (2-tailed) | ,657 | ,216 | ,247 | ,140 | ,800 |

**Test Statistics[a]**

| | BOD-AOD-31 | BOD-AOD-32 | BOD-AOD-33 | BRZ-ARZ-11 | BRZ-ARZ-12 |
|---|---|---|---|---|---|
| Mann-Whitney U | 212,000 | 219,000 | 220,500 | 187,000 | 200,000 |
| Wilcoxon W | 563,000 | 570,000 | 373,500 | 340,000 | 353,000 |
| Z | -,444 | -,089 | -,022 | -1,678 | -,939 |
| Asymp. Sig. (2-tailed) | ,657 | ,929 | ,982 | ,093 | ,348 |

**Test Statistics[a]**

| | BRZ-ARZ-13 | BRZ-ARZ-21 | BRZ-ARZ-22 | BRZ-ARZ-23 | BRZ-ARZ-31 |
|---|---|---|---|---|---|
| Mann-Whitney U | 216,500 | 178,500 | 187,000 | 204,000 | 216,500 |
| Wilcoxon W | 567,500 | 331,500 | 340,000 | 357,000 | 567,500 |
| Z | -,306 | -1,901 | -1,677 | -1,157 | -,306 |
| Asymp. Sig. (2-tailed) | ,759 | ,057 | ,094 | ,247 | ,759 |

Figura A.8: Test de *Mann-Whitney* para diferencias significativas (TB) entre datos sobre errores cometidos por usuarios de diferentes grupos

**Test Statistics[a]**

| | BRZ-ARZ-32 | BRZ-ARZ-33 | BRZ-AOD-11 | BRZ-AOD-12 | BRZ-AOD-13 |
|---|---|---|---|---|---|
| Mann-Whitney U | 208,000 | 204,000 | 221,000 | 217,000 | 208,000 |
| Wilcoxon W | 559,000 | 357,000 | 572,000 | 370,000 | 559,000 |
| Z | -1,237 | -1,157 | ,000 | -,225 | -1,237 |
| Asymp. Sig. (2-tailed) | ,216 | ,247 | 1,000 | ,822 | ,216 |

**Test Statistics[a]**

| | BRZ-AOD-21 | BRZ-AOD-22 | BRZ-AOD-23 | BRZ-AOD-31 |
|---|---|---|---|---|
| Mann-Whitney U | 204,000 | 170,000 | 204,000 | 217,000 |
| Wilcoxon W | 357,000 | 323,000 | 357,000 | 370,000 |
| Z | -1,157 | -2,110 | -1,157 | -,225 |
| Asymp. Sig. (2-tailed) | ,247 | ,035 | ,247 | ,822 |

**Test Statistics[a]**

| | BRZ-AOD-32 | BRZ-AOD-33 |
|---|---|---|
| Mann-Whitney U | 208,500 | 195,500 |
| Wilcoxon W | 361,500 | 348,500 |
| Z | -,617 | -1,434 |
| Asymp. Sig. (2-tailed) | ,537 | ,151 |

a. Grouping Variable: GRP

Figura A.9: Test de *Mann-Whitney* para diferencias significativas (TB) entre datos sobre errores cometidos por usuarios de diferentes grupos

**Test Statistics[d]**

| | BRZ11 - BOD11 | BRZ12 - BOD12 | BRZ13 - BOD13 | BRZ21 - BOD21 | BRZ22 - BOD22 | BRZ23 - BOD23 |
|---|---|---|---|---|---|---|
| Z | -1,000[a] | -,577[a] | ,000[b] | ,000[b] | -1,000[a] | -1,000[a] |
| Asymp. Sig. (2-tailed) | ,317 | ,564 | 1,000 | 1,000 | ,317 | ,317 |

**Test Statistics[d]**

| | BRZ31 - BOD31 | BRZ32 - BOD32 | BRZ33 - BOD33 | AOD11 - ARZ11 | AOD12 - ARZ12 | AOD13 - ARZ13 |
|---|---|---|---|---|---|---|
| Z | -,577[a] | -,816[a] | -1,414[a] | -2,000[a] | -,816[a] | -1,000[a] |
| Asymp. Sig. (2-tailed) | ,564 | ,414 | ,157 | ,046 | ,414 | ,317 |

**Test Statistics[d]**

| | AOD21 - ARZ21 | AOD22 - ARZ22 | AOD23 - ARZ23 | AOD31 - ARZ31 | AOD32 - ARZ32 | AOD33 - ARZ33 |
|---|---|---|---|---|---|---|
| Z | -1,134[a] | -,378[c] | -,378[c] | -,577[c] | -1,732[c] | -,816[c] |
| Asymp. Sig. (2-tailed) | ,257 | ,705 | ,705 | ,564 | ,083 | ,414 |

a. Based on positive ranks.
b. The sum of negative ranks equals the sum of positive ranks.
c. Based on negative ranks.
d. Wilcoxon Signed Ranks Test

Figura A.10: Test de *Wilcoxon* para diferencias significativas (TB) entre datos sobre errores cometidos por usuarios del mismo grupo usando diferentes interfaces

## A.4. Análisis de diferencias significativas en eficiencia

Aquí detallamos el análisis de diferencias significativas según el análisis por grupo-interfaz (GIB) y por tarea (TB). Los test utilizados son el test de *Mann-Whitney* para comparar resultados entre los grupos A y B, y el test de *Wilcoxon* para comparar resultados dentro del mismo grupo pero con diferentes interfaces.

Hemos caracterizado la eficiencia con la siguiente fórmula: $tiempo/(4 - errores)$. Cuanto menor sea el valor de la eficiencia, mejor será el resultado:

- cuanto menor sea el numerador menor tiempo habrán utiliado los usuarios para realizar la tarea.
- cuanto mayor sea el denominador, $(4 - errores)$, menor será el número de errores cometidos. Nótese que el número máximo de errores por cada tarea es de 4.

### A.4.1. Análisis GIB

En la comparativa entre ARZ y AOD, al ignorarse los datos sobre AOD11 y AOD12, tuvimos que ignorar también los datos sobre ARZ11 y ARZ12. En la comparativa entre BOD y BRZ, al ignorarse los datos sobre BOD22, BRZ11 y BRZ12, tuvimos que ignorar también los datos sobre BRZ22, BOD11 y BOD12. El resultado de ambas comparativas se puede ver en la figura A.11.

**Descriptive Statistics**

| | N | Mean | Std. Deviation | Minimum | Maximum |
|---|---|---|---|---|---|
| AOD | 182 | 22,26099 | 16,011929 | ,750 | 81,333 |
| BOD | 102 | 25,72386 | 16,509065 | 3,250 | 69,250 |
| ARZ | 182 | 23,68407 | 20,160852 | 3,500 | 122,500 |
| BRZ | 102 | 19,55719 | 15,241542 | 3,000 | 74,000 |

**Ranks**

| | | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| ARZ - AOD | Negative Ranks | 83[a] | 92,52 | 7679,00 |
| | Positive Ranks | 98[b] | 89,71 | 8792,00 |
| | Ties | 1[c] | | |
| | Total | 182 | | |
| BRZ - BOD | Negative Ranks | 66[d] | 56,44 | 3725,00 |
| | Positive Ranks | 34[e] | 38,97 | 1325,00 |
| | Ties | 2[f] | | |
| | Total | 102 | | |

a. ARZ < AOD
b. ARZ > AOD
c. ARZ = AOD
d. BRZ < BOD
e. BRZ > BOD
f. BRZ = BOD

**Test Statistics[c]**

| | ARZ - AOD | BRZ - BOD |
|---|---|---|
| Z | -,788[a] | -4,126[b] |
| Asymp. Sig. (2-tailed) | ,430 | ,000 |

a. Based on negative ranks.
b. Based on positive ranks.
c. Wilcoxon Signed Ranks Test

Figura A.11: Test de *Wilcoxon* para diferencias significativas (GIB) entre datos sobre el tiempo empleado por usuarios del mismo grupo usando diferentes interfaces

En la comparativa entre AOD y BOD, al ignorarse los datos sobre AOD11, AOD12 y BOD22, tuvimos que ignorar también los datos sobre BOD11, BOD12 y AOD22. El resultado de la comparativa se puede ver en la Fig. A.12

**Descriptive Statistics**

| | N | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|---|
| AOD | 156 | ,750 | 81,333 | 23,86806 | 16,443623 |
| BOD | 102 | 3,250 | 69,250 | 25,72386 | 16,509065 |
| Valid N (listwise) | 102 | | | | |

**Ranks**

| | GROUP | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| AOD vs BOD | AOD | 156 | 126,00 | 19655,50 |
| | BOD | 102 | 134,86 | 13755,50 |
| | Total | 258 | | |

**Test Statistics[a]**

| | AOD vs BOD |
|---|---|
| Mann-Whitney U | 7409,500 |
| Wilcoxon W | 19655,500 |
| Z | -,933 |
| Asymp. Sig. (2-tailed) | ,351 |

a. Grouping Variable: GROUP

Figura A.12: Test de *Mann-Whitney* para diferencias significativas (GIB) entre AOD y BOD

En la comparativa entre AOD y BRZ, no hace falta ignorar más tareas de las anómalas, ya que se ignoran los datos tanto de AOD11 y AOD12, como de BRZ11 y BRZ12. El resultado de la comparativa se puede ver en la Fig. A.13

**Descriptive Statistics**

| | N | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|---|
| AOD | 182 | ,750 | 81,333 | 22,26099 | 16,011929 |
| BRZ | 119 | 3,000 | 74,000 | 17,95238 | 14,711765 |
| Valid N (listwise) | 119 | | | | |

**Ranks**

| | GROUP | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| AOD vs BRZ | 0 | 182 | 161,29 | 29354,50 |
| | 1 | 119 | 135,26 | 16096,50 |
| | Total | 301 | | |

**Test Statistics[a]**

| | AOD vs BRZ |
|---|---|
| Mann-Whitney U | 8956,500 |
| Wilcoxon W | 16096,500 |
| Z | -2,536 |
| Asymp. Sig. (2-tailed) | ,011 |

a. Grouping Variable: GROUP

Figura A.13: Test de *Mann-Whitney* para diferencias significativas (GIB) entre entre AOD y BRZ

En la comparativa entre ARZ y BRZ, al ignorarse los datos sobre BRZ11 y BRZ12, tuvimos que ignorar también los datos sobre ARZ11 y ARZ12. El resultado de la comparativa se puede ver en la Fig. A.14

**Descriptive Statistics**

| | N | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|---|
| ARZ | 182 | 3,500 | 122,500 | 23,68407 | 20,160852 |
| BRZ | 119 | 3,000 | 74,000 | 17,95238 | 14,711765 |
| Valid N (listwise) | 119 | | | | |

**Ranks**

| | GROUP | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| ARZ vs BRZ | ARZ | 182 | 162,09 | 29500,50 |
| | BRZ | 119 | 134,04 | 15950,50 |
| | Total | 301 | | |

**Test Statistics[a]**

| | ARZ vs BRZ |
|---|---|
| Mann-Whitney U | 8810,500 |
| Wilcoxon W | 15950,500 |
| Z | -2,734 |
| Asymp. Sig. (2-tailed) | ,006 |

a. Grouping Variable: GROUP

Figura A.14: Test de *Mann-Whitney* para diferencias significativas (GIB) entre entre ARZ y BRZ

En la comparativa entre ARZ y BOD, al ignorarse los datos sobre BOD22, tuvimos que ignorar también los datos sobre ARZ22. El resultado de la comparativa se puede ver en la Fig. A.15

**Descriptive Statistics**

| | N | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|---|
| BOD | 136 | 3,000 | 105,667 | 27,41238 | 18,679896 |
| ARZ | 208 | 1,500 | 127,667 | 26,03165 | 21,907309 |
| Valid N (listwise) | 136 | | | | |

**Ranks**

| | GROUP | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| BOD vs ARZ | BOD | 136 | 181,89 | 24737,00 |
| | ARZ | 208 | 166,36 | 34603,00 |
| | Total | 344 | | |

**Test Statistics[a]**

| | BOD vs ARZ |
|---|---|
| Mann-Whitney U | 12867,000 |
| Wilcoxon W | 34603,000 |
| Z | -1,416 |
| Asymp. Sig. (2-tailed) | ,157 |

a. Grouping Variable: GROUP

Figura A.15: Test de *Mann-Whitney* para diferencias significativas (GIB) entre entre ARZ y BOD

### A.4.2. Análisis TB

Debido a las anomalías detectadas en las tareas BRZ11 y BRZ12, y que todas las comparaciones son con el grupo BRZ, en general sólo hemos considerado el resto de tareas. Excepto en la comparativa con BOD, que también hemos tenido que ignorar BOD22.

En primer lugar mostramos las estadísticas descriptivas de todas las tareas que hemos tenido en cuenta, véase figura A.16.

**Descriptive Statistics**

| | N | Mean | Std. Deviation | Minimum | Maximum |
|---|---|---|---|---|---|
| AOD13 | 26 | 10,7212 | 8,76565 | 1,00 | 36,50 |
| AOD21 | 26 | 29,8173 | 15,91456 | 4,75 | 81,33 |
| AOD22 | 26 | 12,6186 | 8,18285 | 3,00 | 30,25 |
| AOD23 | 26 | 14,8782 | 6,61871 | 4,50 | 29,33 |
| AOD31 | 26 | 45,4199 | 11,45679 | 26,50 | 71,00 |
| AOD32 | 26 | 20,7468 | 14,37636 | 7,75 | 55,75 |
| AOD33 | 26 | 21,6250 | 12,86628 | ,75 | 53,00 |
| ARZ13 | 26 | 7,8494 | 3,94462 | 3,50 | 21,33 |
| ARZ21 | 26 | 33,4135 | 12,56851 | 16,25 | 73,00 |
| ARZ22 | 26 | 14,8654 | 16,20967 | 5,25 | 91,00 |
| ARZ23 | 26 | 14,5321 | 7,23913 | 5,50 | 32,33 |
| ARZ31 | 26 | 53,2019 | 26,49189 | 19,50 | 122,50 |
| ARZ32 | 26 | 19,5385 | 8,07053 | 4,00 | 34,00 |
| ARZ33 | 26 | 22,3878 | 15,30192 | 5,50 | 65,00 |
| BOD13 | 17 | 10,7941 | 5,03677 | 3,25 | 21,00 |
| BOD21 | 17 | 33,3676 | 12,34343 | 12,50 | 58,75 |
| BOD23 | 17 | 16,4706 | 6,07240 | 5,00 | 25,50 |
| BOD31 | 17 | 51,6373 | 8,88775 | 33,75 | 69,25 |
| BOD32 | 17 | 23,6176 | 12,98380 | 7,50 | 46,50 |
| BOD33 | 17 | 18,4559 | 9,70047 | 8,00 | 43,25 |
| BRZ13 | 17 | 8,9412 | 3,53391 | 3,75 | 15,00 |
| BRZ21 | 17 | 24,0000 | 6,68019 | 15,50 | 38,50 |
| BRZ22 | 17 | 8,3235 | 3,85556 | 4,50 | 16,75 |
| BRZ23 | 17 | 9,7206 | 5,98493 | 5,50 | 30,75 |
| BRZ31 | 17 | 40,1716 | 20,32722 | 19,00 | 74,00 |
| BRZ32 | 17 | 14,2745 | 5,79195 | 4,00 | 29,50 |
| BRZ33 | 17 | 20,2353 | 14,19768 | 3,00 | 50,50 |

Figura A.16: Estadisticas descriptivas de las tareas consideradas en el análisis TB de eficiencia

Salvo dos excepciones, los datos con los que trabajamos tienen una distribución normal, por ello usaremos el test $t$ de Student para calcular las diferencias significativas en cuanto a eficiencia.

En la figura A.17 mostramos el análisis BOD-BRZ, que estudia el cambio de interfaz junto con el aumento de la experiencia con las tareas.

En la figura A.18 mostramos el análisis ARZ-BRZ, que estudia el aumento de la experiencia en las tareas realizadas con *R-Zoom*.

**Paired Samples Test**

| | | Paired Differences | | | | | t |
|---|---|---|---|---|---|---|---|
| | | Mean | Std. Deviation | Std. Error Mean | 95% Confidence Interval of the Difference Lower | Upper | |
| Pair 1 | BOD13 - BRZ13 | 1,85294 | 5,65068 | 1,37049 | -1,05237 | 4,75825 | 1,352 |
| Pair 2 | BOD21 - BRZ21 | 9,36765 | 12,43204 | 3,01521 | 2,97568 | 15,75961 | 3,107 |
| Pair 3 | BOD23 - BRZ23 | 6,75000 | 8,29768 | 2,01248 | 2,48372 | 11,01628 | 3,354 |
| Pair 4 | BOD31 - BRZ31 | 11,46569 | 21,92134 | 5,31671 | ,19477 | 22,73660 | 2,157 |
| Pair 5 | BOD32 - BRZ32 | 9,34314 | 15,14622 | 3,67350 | 1,55567 | 17,13060 | 2,543 |
| Pair 6 | BOD33 - BRZ33 | -1,77941 | 16,81292 | 4,07773 | -10,42382 | 6,86500 | -,436 |

**Paired Samples Test**

| | | df | Sig. (2-tailed) |
|---|---|---|---|
| Pair 1 | BOD13 - BRZ13 | 16 | ,195 |
| Pair 2 | BOD21 - BRZ21 | 16 | ,007 |
| Pair 3 | BOD23 - BRZ23 | 16 | ,004 |
| Pair 4 | BOD31 - BRZ31 | 16 | ,047 |
| Pair 5 | BOD32 - BRZ32 | 16 | ,022 |
| Pair 6 | BOD33 - BRZ33 | 16 | ,668 |

Figura A.17: Análisis de eficiencia BOD-BRZ

**Independent Samples Test**

| | | Levene's Test for Equality of Variances | | t-test for Equality of Means | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | F | Sig. | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference | 95% Confidence Interval of the Difference Lower | Upper |
| ARZ_BRZ_21 | Equal variances assumed | 3,414 | ,072 | 2,830 | 41 | ,007 | 9,41346 | 3,32639 | 2,69569 | 16,13123 |
| | Equal variances not assumed | | | 3,191 | 39,692 | ,003 | 9,41346 | 2,94969 | 3,45047 | 15,37645 |
| ARZ_BRZ_23 | Equal variances assumed | 2,065 | ,158 | 2,276 | 41 | ,028 | 4,81146 | 2,11390 | ,54236 | 9,08056 |
| | Equal variances not assumed | | | 2,370 | 38,629 | ,023 | 4,81146 | 2,03042 | ,70329 | 8,91963 |
| ARZ_BRZ_31 | Equal variances assumed | ,227 | ,636 | 1,721 | 41 | ,093 | 13,03035 | 7,57093 | -2,25945 | 28,32016 |
| | Equal variances not assumed | | | 1,819 | 39,831 | ,076 | 13,03035 | 7,16231 | -1,44713 | 27,50784 |
| ARZ_BRZ_32 | Equal variances assumed | 3,956 | ,053 | 2,322 | 41 | ,025 | 5,26395 | 2,26657 | ,68653 | 9,84138 |
| | Equal variances not assumed | | | 2,487 | 40,567 | ,017 | 5,26395 | 2,11624 | ,98874 | 9,53917 |
| ARZ_BRZ_33 | Equal variances assumed | ,154 | ,697 | ,464 | 41 | ,645 | 2,15253 | 4,64139 | -7,22095 | 11,52600 |
| | Equal variances not assumed | | | ,471 | 36,178 | ,640 | 2,15253 | 4,56761 | -7,10943 | 11,41448 |

**Ranks**

| | GROUP | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| ARZ_BRZ_22 | ARZ | 17 | 15,71 | 267,00 |
| | BRZ | 26 | 26,12 | 679,00 |
| | Total | 43 | | |

**Test Statistics[a]**

| | ARZ_BRZ_22 |
|---|---|
| Mann-Whitney U | 114,000 |
| Wilcoxon W | 267,000 |
| Z | -2,659 |
| Asymp. Sig. (2-tailed) | ,008 |

a. Grouping Variable: GROUP

Figura A.18: Análisis de eficiencia ARZ-BRZ

En la figura A.19 mostramos el análisis AOD-BRZ, que estudia el cambio de interfaz en usuarios experimentados.

**Independent Samples Test**

| | | Levene's Test for Equality of Variances | | t-test for Equality of Means | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | 95% Confidence Interval of the Difference | |
| | | F | Sig. | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference | Lower | Upper |
| BRZ_AOD_13 | Equal variances assumed | 5,470 | ,024 | ,793 | 41 | ,432 | 1,77998 | 2,24323 | -2,75032 | 6,31027 |
| | Equal variances not assumed | | | ,927 | 35,542 | ,360 | 1,77998 | 1,92090 | -2,11754 | 5,67749 |
| BRZ_AOD_21 | Equal variances assumed | 6,338 | ,016 | 1,423 | 41 | ,162 | 5,81731 | 4,08881 | -2,44022 | 14,07484 |
| | Equal variances not assumed | | | 1,654 | 36,184 | ,107 | 5,81731 | 3,51657 | -1,31338 | 12,94799 |
| BRZ_AOD_22 | Equal variances assumed | 11,183 | ,002 | 2,017 | 41 | ,050 | 4,29506 | 2,12988 | -,00632 | 8,59644 |
| | Equal variances not assumed | | | 2,312 | 38,012 | ,026 | 4,29506 | 1,85736 | ,53507 | 8,05505 |
| BRZ_AOD_23 | Equal variances assumed | ,995 | ,324 | 2,592 | 41 | ,013 | 5,15762 | 1,98961 | 1,13952 | 9,17571 |
| | Equal variances not assumed | | | 2,649 | 36,772 | ,012 | 5,15762 | 1,94729 | 1,21121 | 9,10402 |
| BRZ_AOD_31 | Equal variances assumed | 11,697 | ,001 | 1,083 | 41 | ,285 | 5,24830 | 4,84491 | -4,53620 | 15,03280 |
| | Equal variances not assumed | | | ,969 | 22,710 | ,343 | 5,24830 | 5,41793 | -5,96748 | 16,46409 |
| BRZ_AOD_33 | Equal variances assumed | ,103 | ,749 | ,332 | 41 | ,741 | 1,38971 | 4,18003 | -7,05204 | 9,83145 |
| | Equal variances not assumed | | | ,326 | 31,908 | ,747 | 1,38971 | 4,26899 | -7,30692 | 10,08633 |

**Ranks**

| | GROUP | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| BRZ_AOD_32 | BRZ | 17 | 20,29 | 345,00 |
| | AOD | 26 | 23,12 | 601,00 |
| | Total | 43 | | |

**Test Statistics[a]**

| | BRZ_AOD_32 |
|---|---|
| Mann-Whitney U | 192,000 |
| Wilcoxon W | 345,000 |
| Z | -,721 |
| Asymp. Sig. (2-tailed) | ,471 |

a. Grouping Variable: GROUP

Figura A.19: Análisis de eficiencia AOD-BRZ

## A.5. Análisis de los comentarios abiertos sobre las interfaces

A continuación enumeramos los comentarios (véase la tabla A.2) proporcionando el porcentaje de usuarios que han hecho ese comentario, si es una ventaja o un inconveniente, y una breve descripción del comentario realizado.

| % | V/I | Descripción del comentario |
|---|---|---|
| 40 % | Ventaja | R-Zoom ofrece una visión global de la colección de miniaturas. |
| 24 % | Ventaja | R-Zoom muestra las vistas global y detallada en la misma ventana. |
| 20 % | Inconveniente | O+D muestra las vistas global y detallada en distintas ventanas. |
| 17,1 % | Ventaja | O+D ofrece una visión global de la colección de miniaturas. |
| 17,1 % | Ventaja | R-Zoom permite orientarse al usuario. |
| 14,3 % | Ventaja | R-Zoom permite moverse con los cursores. |
| 11,4 % | Inconveniente | Con O+D es fácil que el usuario se pierda. |
| 8,6 % | Ventaja | Con R-Zoom se identifica fácilmente el foco. |
| 8,6 % | Inconveniente | En ocasiones, en R-Zoom el aumento aplicado al foco es algo menor que en O+D. |
| 6,7 % | Ventaja | Ambas interfaces muestran una buena visualización de los pasos de una evaluación. |
| 6,7 % | Inconveniente | Con O+D es más difícil realizar las búsquedas de visualizaciones. |
| 6,7 % | Inconveniente | Con O+D, si el número de visualizaciones es alto la visibilidad es poca. |
| 2,9 % | Ventaja | R-Zoom permite manejar muchas visualizaciones a la vez. |
| 2,9 % | Inconveniente | Con O+D hay que cambiar el punto de atención constantemente. |
| 2,9 % | Inconveniente | Con O+D se solapan las pantallas de vista global y detallada. |

Tabla A.2: Listado de comentarios libres realizados por los usuarios. Muestra el % de usuarios que ha realizado el comentario, si se identificó como una ventaja o un inconveniente y una breve descripción del comentario

# Apéndice B

# Formato XML de las animaciones web

## B.1. DTD, información de mantenimiento

```
<!ELEMENT principal (content,style,animation)>
<!ELEMENT content (page)>
<!ELEMENT page (title,MainHead,ProblemDescription,AlgorithmDescription,ProgramCode)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT MainHead (#PCDATA)>
    <!ELEMENT ProblemDescription (PHead,PText)>
        <!ELEMENT PHead      (#PCDATA)>
        <!ELEMENT PText      (#PCDATA)>
    <!ELEMENT AlgorithmDescription (AHead,AText)>
        <!ELEMENT AHead      (#PCDATA)>
        <!ELEMENT AText      (#PCDATA)>
    <!ELEMENT ProgramCode (CHead,CText)>
        <!ELEMENT CHead      (#PCDATA)>
        <!ELEMENT CText      (#PCDATA)>
    <!ENTITY content SYSTEM "contenido.xml">
<!ELEMENT style (root)>
<!ELEMENT root (MainH,ProblemD,AlgorithmD,ProgramC)>
    <!ELEMENT MainH (size,align,color)>
        <!ELEMENT size       (#PCDATA)>
        <!ELEMENT align      (#PCDATA)>
        <!ELEMENT color          (#PCDATA)>
    <!ELEMENT ProblemD (PH)>
        <!ELEMENT PH (size,align,color)>
    <!ELEMENT AlgorithmD (AH)>
        <!ELEMENT AH (size,align,color)>
    <!ELEMENT ProgramC (CH)>
        <!ELEMENT CH (size,align,color)>
    <!ENTITY style SYSTEM "estilo.xml">
<!ELEMENT animation (expression,photogramslist,surconfig)>
<!--expresion(expression),lista de fotogramas(photogramslist),configuracion de entorno(surconfig)-->
    <!ELEMENT expression            (#PCDATA)>
    <!ELEMENT photogramslist        (#PCDATA)>
    <!ELEMENT surconfig         (#PCDATA)>
```

## B.2. Ejemplo de una animación web

```
<?XML version="1.0"?>
<!DOCTYPE animcontent SYSTEM "anim.dtd">
<principal>
  <content>
    <page>
      <title>Mirror</title>
      <MainHead>Mirror</MainHead>
      <ProblemDescription>
        <PHead>Problem description</PHead>
        <PText>
          Given a binary tree, compute another tree which is the result of mirroring the first one.
          A mirrored tree is a tree of the same depth and the same number of nodes.
          Each node has the same parent node and the same children nodes.
          If a node is the leftmost of its brothers, its mirrored version will be the rightmost one.
          If a node is the rightmost of its brothers, its mirrored version will be the leftmost one.
          Given two brothers nodes, if a node "x" is on the left side of a node "y", then their
          mirrored version will be swapped.
        </PText>
      </ProblemDescription>
      <AlgorithmDescription>
        <AHead>Algorithm description</AHead>
        <AText>
          The solution consists in swapping every children nodes in the tree.
          We will use a recursive function that swaps children nodes, until this nodes are empty
          nodes.
        </AText>
      </AlgorithmDescription>
      <ProgramCode>
        <CHead>Source code</CHead>
        <CText>
! Function mirror, compute the symmetric tree of a given one
! -------------------------------------------------------------
data tree == empty ++ node (tree X num X tree);
dec mirror: tree -> tree;
--- mirror (empty) <= empty;
--- mirror node (hi, n, hd) <= node (mirror (hd), n, mirror (hi));
        </CText>
      </ProgramCode>
    </page>
  </content>
  <style>
    <root>
      <MainH><size>6</size> <align>center</align> <color>black</color></MainH>
      <ProblemD><PH><size>2</size> <align>left</align> <color>black</color></PH></ProblemD>
      <AlgorithmD><AH><size>2</size> <align>left</align> <color>black</color></AH></AlgorithmD>
      <ProgramC><CH><size>2</size> <align>left</align> <color>black</color></CH></ProgramC>
      &style;
    </root>
  </style>
  <animation>
      <expression>
        mirror (node(node(empty,1,node(empty,8,empty)),0,
        node(empty,2,node(node(empty,65,empty),9,empty))));
```

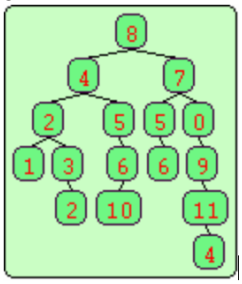
```
      </expression>
       <photogramslist>1 2 3 4 5 6 8 10 11 14</photogramslist>
       <surconfig>
         <!--Contents of the .ini file. Ignored to be lots of data-->
       </surconfig>
  </animation>
</principal>
```

# Apéndice C

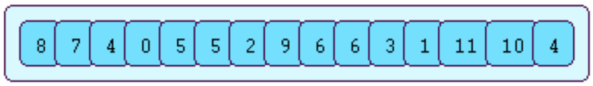
# Detalles de la evaluación a corto plazo de la animaciones web

## C.1. Test de conocimientos sobre el algoritmo

1. En su opinión, ¿cuál es la idea básica que utiliza el algoritmo?
2. ¿Cuál es el resultado del algoritmo ante un árbol como el siguiente?




3. Dada la siguiente salida, resultado de aplicar el algoritmo a un árbol:




   Dibuje el árbol de entrada, sabiendo que todo nodo interior tiene al menos dos nodos hijos.

4. ¿Qué relación deberían cumplir los nodos del árbol de entrada para que la lista resultante esté ordenada en orden estrictamente creciente?
5. El algoritmo de recorrido en anchura descrito implementa el recorrido de izquierda a derecha. Explique los cambios necesarios en el algoritmo para hacer un recorrido, también en anchura pero de derecha a izquierda:

## C.2. Cuestionarios de opinión de los estudiantes

### C.2.1. Cuestionario para los estudiantes del grupo de visores (GV)

1. ¿Cuántas veces, antes de hoy, has usado las animaciones web?

   a) Nunca.

   b) Sólo en la clase anterior.

   c) En la clase anterior y alguna vez más, por curiosidad.

   d) Frecuentemente.

   e) Siempre que he estudiado algún algoritmo con animaciónes web disponibles.

2. El uso de las animaciones web me ha ayudado a comprender mejor el algoritmo.

   a) Totalmente de acuerdo.

   b) De acuerdo.

   c) Sin opinión.

   d) En desacuerdo.

   e) Totalmente en desacuerdo.

3. ¿Qué crees que ayuda más a comprender el algoritmo?

   a) Generar las animaciones ayuda más que solamente usarlas.

   b) Usar las animaciones ayuda más que generarlas.

   c) Ambas ayudan lo mismo, pero creo que deberían generarse y usarse.

   d) Ambas ayudan lo mismo.

   e) Sin Opinión.

4. En general, las animaciones web son fáciles de usar.

   a) Totalmente de acuerdo.

   b) De acuerdo.

   c) Sin opinión.

   d) En desacuerdo.

   e) Totalmente en desacuerdo.

5. En general, creo que las animaciones web son útiles.

   a) Totalmente de acuerdo.

   b) De acuerdo.

   c) Sin opinión.

   d) En desacuerdo.

   e) Totalmente en desacuerdo.

6. Creo que la disponibilidad de varios formatos para ver la animación web es útil.

   a) Totalmente de acuerdo.

   b) De acuerdo.

   c) Sin opinión.

*d*) En desacuerdo.

*e*) Totalmente en desacuerdo.

7. De los cuatro componentes de una animación web: descripción del problema, descripción de la animación, código fuente y animación; seleccione aquellos que cree que sobran.

   *a*) Descripción del problema.

   *b*) Descripción del algoritmo.

   *c*) Código fuente.

   *d*) Animación.

8. ¿Añadiría algo más a una animación web?

9. Enumere los aspectos que cree más **negativos** de una animación web.

10. Enumere los aspectos que cree más **positivos** de una animación web.

### C.2.2. Cuestionario para los estudiantes del grupo de constructores (GC)

1. ¿Cuántas veces, antes de hoy, has creado las animaciones web?

   *a*) Nunca.

   *b*) Sólo en la clase anterior.

   *c*) En la clase anterior y alguna vez más, por curiosidad.

   *d*) Frecuentemente.

   *e*) Siempre que he estudiado algún algoritmo con animaciónes web disponibles.

2. Construir animaciones web me ha ayudado a comprender mejor el algoritmo.

   *a*) Totalmente de acuerdo.

   *b*) De acuerdo.

   *c*) Sin opinión.

   *d*) En desacuerdo.

   *e*) Totalmente en desacuerdo.

3. ¿Qué crees que ayuda más a comprender el algoritmo?

   *a*) Generar las animaciones ayuda más que solamente usarlas.

   *b*) Usar las animaciones ayuda más que generarlas.

   *c*) Ambas ayudan lo mismo, pero creo que deberían generarse y usarse.

   *d*) Ambas ayudan lo mismo.

   *e*) Sin Opinión.

4. En general, las animaciones web son fáciles de construir con WinHIPE.

   *a*) Totalmente de acuerdo.

*b*) De acuerdo.

*c*) Sin opinión.

*d*) En desacuerdo.

*e*) Totalmente en desacuerdo.

5. Una herramienta para gestionar colecciones de animaciones web me parecería útil.

   *a*) Totalmente de acuerdo.

   *b*) De acuerdo.

   *c*) Sin opinión.

   *d*) En desacuerdo.

   *e*) Totalmente en desacuerdo.

6. Creo que la disponibilidad de varios formatos para ver la animación web es útil.

   *a*) Totalmente de acuerdo.

   *b*) De acuerdo.

   *c*) Sin opinión.

   *d*) En desacuerdo.

   *e*) Totalmente en desacuerdo.

7. Enumere los aspectos que cree más **negativos** de una animación web.

8. Enumere los aspectos que cree más **positivos** de una animación web.

## C.3. Descripción textual del algoritmo de recorrido de árboles en anchura

### Descripción del problema

Sea un árbol binario b. Se desea obtener una lista con los elementos contenidos en sus nodos. Si nos imaginamos el árbol dibujado de la forma convencional, es decir, con la raíz arriba, la lista resultante debe contener los elementos del árbol en orden descendente de nivel. Es decir, en primer lugar irá el elemento contenido en la raíz, después el nodo contenido en la raíz del hijo izquierdo, luego la del hijo derecho, y así sucesivamente.

### Descripción del algoritmo

El problema planteado no tiene una solución trivial. Por un lado deberá resolverse mediante un proceso repetitivo que recorra todos los nodos del árbol pero, por otro, la estructura recursiva del árbol no sirve como estructura de un proceso recursivo.

Si probamos con varios ejemplos, podemos ver que al ir tomando elementos para el resultado, va quedando pendiente tomar elementos de varios subárboles. Por tanto, conviene disponer de una función auxiliar que genere la lista de elementos pedida a partir de una lista de árboles.

La idea básica del algoritmo es la siguiente. Inicialmente el árbol se incluye en una lista y se llama a la función auxiliar. Esta función auxiliar será recursiva porque realizará todo el proceso repetitivo necesario. Cuando la función se encuentre con una lista vacía de árboles, se acaba el proceso. Si la función encuentra una lista no vacía cuyo primer elemento es un árbol vacío, quiere decir que ese subárbol podemos descartarlo y continuar. El caso más general surgirá cuando tengamos una lista no vacía y cuyo elemento de cabeza sea un árbol no vacío. En ese caso, se toma el elemento de la raíz como parte del resultado y el resultado (recursivo) para sus dos hijos corresponde a un nivel inferior, por lo que se tendrá que calcular al final de la lista de árboles pendientes.

## C.4. Detalles estadísticos de la evaluación a corto plazo

### C.4.1. Análisis de normalidad de los datos

**One-Sample Kolmogorov-Smirnov Test**

| | | TPOAPREND | TPOTEST | P11 | P12 |
|---|---|---|---|---|---|
| N | | 13 | 13 | 13 | 13 |
| Normal Parameters[a,b] | Mean | 32,3077 | 17,3846 | ,6154 | ,3846 |
| | Std. Deviation | 16,84431 | 5,70874 | ,50637 | ,50637 |
| Most Extreme Differences | Absolute | ,264 | ,165 | ,392 | ,392 |
| | Positive | ,264 | ,104 | ,272 | ,392 |
| | Negative | -,238 | -,165 | -,392 | -,272 |
| Kolmogorov-Smirnov Z | | ,951 | ,597 | 1,412 | 1,412 |
| Asymp. Sig. (2-tailed) | | ,326 | ,869 | ,037 | ,037 |

| | | P13 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| N | | 13 | 13 | 13 | 13 |
| Normal Parameters[a,b] | Mean | ,3077 | 16,0000 | 14,3846 | 1,2846 |
| | Std. Deviation | ,48038 | ,00000[c] | 2,21880 | 1,74492 |
| Most Extreme Differences | Absolute | ,431 | | ,532 | ,488 |
| | Positive | ,431 | | ,391 | ,488 |
| | Negative | -,261 | | -,532 | -,250 |
| Kolmogorov-Smirnov Z | | 1,555 | | 1,919 | 1,759 |
| Asymp. Sig. (2-tailed) | | ,016 | | ,001 | ,004 |

| | | P5 |
|---|---|---|
| N | | 13 |
| Normal Parameters[a,b] | Mean | ,5308 |
| | Std. Deviation | ,33011 |
| Most Extreme Differences | Absolute | ,373 |
| | Positive | ,373 |
| | Negative | -,242 |
| Kolmogorov-Smirnov Z | | 1,345 |
| Asymp. Sig. (2-tailed) | | ,054 |

a. Test distribution is Normal.

b. Calculated from data.

c. The distribution has no variance for this variable. One-Sample Kolmogorov-Smirnov Test cannot be performed.

Figura C.1: Test de normalidad para los datos obtenidos. TPOAPREND y TPOTEST, representan respectivamente a los tiempos empleados en estudiar el algoritmo y responder al test de conocimientos. P11, P12 y P13, representan la identificación de las tres ideas básicas del algoritmo

### C.4.2. Análisis de las respuestas al test de conocimientos

**Test Statistics[b]**

| | P11 | P12 | P13 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|---|---|
| Mann-Whitney U | 12,500 | 10,000 | 7,000 | 21,000 | 18,000 | 13,500 | 8,000 |
| Wilcoxon W | 40,500 | 38,000 | 35,000 | 49,000 | 46,000 | 41,500 | 36,000 |
| Z | -1,437 | -1,859 | -2,494 | ,000 | -,926 | -1,224 | -2,156 |
| Asymp. Sig. (2-tailed) | ,151 | ,063 | ,013 | 1,000 | ,355 | ,221 | ,031 |
| Exact Sig. [2*(1-tailed Sig.)] | ,234[a] | ,138[a] | ,051[a] | 1,000[a] | ,731[a] | ,295[a] | ,073[a] |

a. Not corrected for ties.

b. Grouping Variable: GRUPONUM

**Group Statistics**

| | GRUPO | N | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|
| P5 | GC | 6 | ,7667 | ,36148 | ,14757 |
| | GV | 7 | ,3286 | ,07559 | ,02857 |

**Independent Samples Test**

| | | Levene's Test for Equality of Variances | |
|---|---|---|---|
| | | F | Sig. |
| P5 | Equal variances assumed | 27,107 | ,000 |
| | Equal variances not assumed | | |

| | | t-test for Equality of Means | | | |
|---|---|---|---|---|---|
| | | t | df | Sig. (2-tailed) | Mean Difference |
| P5 | Equal variances assumed | 3,150 | 11 | ,009 | ,43810 |
| | Equal variances not assumed | 2,915 | 5,376 | ,030 | ,43810 |

Figura C.2: Test de diferencias significativas para las respuestas al test de conocimientos. Como los datos de las respuestas a la pregunta 5 (P5) se encuentran en la frontera de normalidad en cuanto al tipo de distrbución, hemos analizado sus diferencias con dos tests: $t$ de Student y Wilcoxon

### C.4.3. Análisis de tiempos

**Group Statistics**

| | GRUPO | N | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|
| TPOAPREND | GC | 6 | 49,0000 | 4,97996 | 2,03306 |
| | GV | 7 | 18,0000 | 5,41603 | 2,04707 |
| TPOTEST | GC | 6 | 19,6667 | 5,12510 | 2,09231 |
| | GV | 7 | 15,4286 | 5,79819 | 2,19151 |

**Independent Samples Test**

| | | Levene's Test for Equality of Variances | |
|---|---|---|---|
| | | F | Sig. |
| TPOAPREND | Equal variances assumed | ,002 | ,964 |
| | Equal variances not assumed | | |
| TPOTEST | Equal variances assumed | ,262 | ,619 |
| | Equal variances not assumed | | |

| | | t-test for Equality of Means | | | |
|---|---|---|---|---|---|
| | | t | df | Sig. (2-tailed) | Mean Difference |
| TPOAPREND | Equal variances assumed | 10,670 | 11 | ,000 | 31,00000 |
| | Equal variances not assumed | 10,745 | 10,922 | ,000 | 31,00000 |
| TPOTEST | Equal variances assumed | 1,384 | 11 | ,194 | 4,23810 |
| | Equal variances not assumed | 1,399 | 10,978 | ,190 | 4,23810 |

Figura C.3: Test de diferencias significativas para tiempos

# Apéndice D

# Detalles de la evaluación a largo plazo de la animaciones web

## D.1. Examen sobre programación funcional

El examen de la asignatura sobre programación funcional se dividía en tres preguntas: práctica obligatoria, cuestión y problema.

### Enunciado de la práctica obligatoria

El siguiente programa HOPE devuelve el cambio de una compra especificando el número de billetes de 10 y 5 euros, y el número de monedas de 2 y 1 euro:

```
dec cambio : num # num -> num # num # num # num;
--- cambio (pagado,precio) <=
  if pagado =< precio then (0,0,0,0)
  else
        ((pagado-precio) div 10 ,
         (pagado-precio) mod 10 div 5 ,
         (pagado-precio) mod 10 mod 5 div 2 ,
         (pagado-precio) mod 10 mod 5 mod 2);
```

Se pide optimizar la función `cambio` utilizando definiciones locales.

### Enunciado de la cuestión

La función `FuncionX` se define de la siguiente manera:

```
dec FuncionX : list(real) -> real;
--- FuncionX(L)<=FuncionY(L,L);
dec FuncionY : list(real)#list(real) -> real;
--- FuncionY (L,e::R)<=e/FuncionZ(L) +FuncionY (L,R);
--- FuncionY(_, nil)<= 0.0;
```

```
dec FuncionZ : list(real) -> real;
--- FuncionZ (e::R)<=1.0 + FuncionZ(R);
--- FuncionZ(nil)<=0.0;
```

¿Qué valor devuelven las siguientes expresiones? Detalla la ejecución simulada de las mismas.

```
FuncionX [1.0 2.0 3.0 4.0 5.0]
FuncionX [2.0 2.0 10.0 10.0]
```

### Enunciado del problema

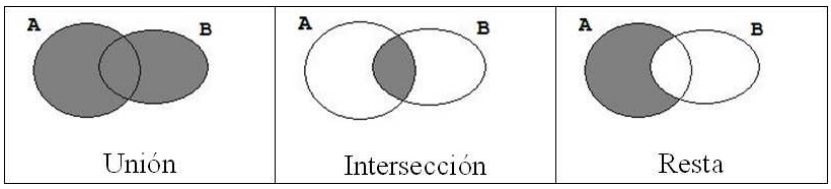
Codificar en HOPE un tipo de datos que permita manipular conjuntos de números enteros. El tipo de datos se llamará CONJUNTO, y no se podrá definir utilizando las listas predefinidas en HOPE. Un conjunto de este tipo permite almacenar un número indeterminado de números (todos distintos), incluso ninguno (conjunto vacío).

Además se pide codificar las siguientes funciones y operadores:

**Cardinal** una función que devolverá el número de elementos que tiene un conjunto.

**Pertenece** una función que devolverá `true` si el número pasado como parámetro se encuentra dentro del conjunto, también pasado como parámetro.

**Operadores infijos** unión, intersección y resta. A continuación se muestran los esquemas que describen cada operación, el resultado se encuentra sombreado.




La unión tendrá menor prioridad que la intersección. La resta tendrá la misma prioridad que la unión. Nótese que la operación de la resta "a menos b" (cuyo resultado se muestra en la figura) no es conmutativa.

## D.2. Cuestionarios de opinión de los estudiantes sobre las animaciones web

### D.2.1. Preguntas realizadas a los estudiantes del grupo GV

Pregunta 1. Creo que el uso de las animaciones web con WinHIPE me ha ayudado a comprender los conceptos explicados:

| | | | |
|---|---|---|---|
| a) Totalmente de acuerdo | b) De acuerdo | c) En desacuerdo | d) Totalmente en desacuerdo |

Pregunta 2. Creo que las animaciones web son fáciles de usar:

| | | | |
|---|---|---|---|
| a) Totalmente de acuerdo | b) De acuerdo | c) En desacuerdo | d) Totalmente en desacuerdo |

Pregunta 3. Creo que las animaciones web son útiles:

| | | | |
|---|---|---|---|
| a) Totalmente de acuerdo | b) De acuerdo | c) En desacuerdo | d) Totalmente en desacuerdo |

Pregunta 4. Creo que la posibilidad de ver las animaciones web en varios formatos distintos es útil:

| | | | |
|---|---|---|---|
| a) Totalmente de acuerdo | b) De acuerdo | c) En desacuerdo | d) Totalmente en desacuerdo |

Pregunta 5. ¿Qué cree usted que es mejor para el aprendizaje?
a) Construir animaciones es mejor que verlas.
b) Ver las animaciones es mejor que construirlas.
c) Ambas son iguales pero deberían usarse conjuntamente.
d) Ambas son iguales, cualquiera vale.

### D.2.2. Preguntas realizadas a los estudiantes del grupo GC

Pregunta 1. Creo que la construcción de las animaciones web con WinHIPE me ha ayudado a comprender los conceptos explicados:

| | | | |
|---|---|---|---|
| a) Totalmente de acuerdo | b) De acuerdo | c) En desacuerdo | d) Totalmente en desacuerdo |

Pregunta 2. Creo que las animaciones web son fáciles de construir con WinHIPE:

| | | | |
|---|---|---|---|
| a) Totalmente de acuerdo | b) De acuerdo | c) En desacuerdo | d) Totalmente en desacuerdo |

Pregunta 3. Creo que las animaciones web son útiles:

| | | | |
|---|---|---|---|
| a) Totalmente de acuerdo | b) De acuerdo | c) En desacuerdo | d) Totalmente en desacuerdo |

Pregunta 4. Creo que la posibilidad de ver las animaciones web en varios formatos distintos es útil:

| | | | |
|---|---|---|---|
| a) Totalmente de acuerdo | b) De acuerdo | c) En desacuerdo | d) Totalmente en desacuerdo |

Pregunta 5. ¿Qué cree usted que es mejor para el aprendizaje?
a) Construir animaciones es mejor que verlas.
b) Ver las animaciones es mejor que construirlas.
c) Ambas son iguales pero deberían usarse conjuntamente.
d) Ambas son iguales, cualquiera vale.

## D.3. Detalles estadísticos de la evaluación

### D.3.1. Presentación de los estudiantes al examen

**Resumen del procesamiento de los casos**

| | Casos | | | | | |
|---|---|---|---|---|---|---|
| | Válidos | | Perdidos | | Total | |
| | N | Porcentaje | N | Porcentaje | N | Porcentaje |
| GRUPO * Presentado | 388 | 67,7% | 185 | 32,3% | 573 | 100,0% |

**Tabla de contingencia GRUPO * Presentado**

| | | | Presentado | | |
|---|---|---|---|---|---|
| | | | 0 | 1 | Total |
| GRUPO | GC | Recuento | 41 | 78 | 119 |
| | | % de GRUPO | 34,5% | 65,5% | 100,0% |
| | GV | Recuento | 47 | 73 | 120 |
| | | % de GRUPO | 39,2% | 60,8% | 100,0% |
| | GT | Recuento | 59 | 90 | 149 |
| | | % de GRUPO | 39,6% | 60,4% | 100,0% |
| Total | | Recuento | 147 | 241 | 388 |
| | | % de GRUPO | 37,9% | 62,1% | 100,0% |

**Pruebas de chi-cuadrado**

| | Valor | gl | Sig. asintótica (bilateral) |
|---|---|---|---|
| Chi-cuadrado de Pearson | ,865 | 2 | ,649 |
| N de casos válidos | 388 | | |

Figura D.1: Análisis de datos de presentación al examen por parte de todos los estudiantes matriculados

**Case Processing Summary**

| | Cases | | | | | |
|---|---|---|---|---|---|---|
| | Valid | | Missing | | Total | |
| | N | Percent | N | Percent | N | Percent |
| GRUPO * Presentado | 167 | 29,1% | 406 | 70,9% | 573 | 100,0% |

**GRUPO * Presentado Crosstabulation**

| | | | Presentado | | |
|---|---|---|---|---|---|
| | | | 0 | 1 | Total |
| GRUPO | GC | Count | 9 | 47 | 56 |
| | | % within GRUPO | 16,1% | 83,9% | 100,0% |
| | GV | Count | 18 | 39 | 57 |
| | | % within GRUPO | 31,6% | 68,4% | 100,0% |
| | GT | Count | 19 | 35 | 54 |
| | | % within GRUPO | 35,2% | 64,8% | 100,0% |
| Total | | Count | 46 | 121 | 167 |
| | | % within GRUPO | 27,5% | 72,5% | 100,0% |

**Chi-Square Tests**

| | Value | df | Asymp. Sig. (2-sided) |
|---|---|---|---|
| Pearson Chi-Square | 5,738 | 2 | ,057 |
| N of Valid Cases | 167 | | |

Figura D.2: Análisis de datos de presentación al examen por parte de los participantes en la evaluación

**Ranks**

| | GRUPO | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| Presentado | GC | 56 | 60,66 | 3397,00 |
| | GT | 54 | 50,15 | 2708,00 |
| | Total | 110 | | |

**Test Statistics[a]**

| | Presentado |
|---|---|
| Mann-Whitney U | 1223,000 |
| Wilcoxon W | 2708,000 |
| Z | -2,290 |
| Asymp. Sig. (2-tailed) | ,022 |

a. Grouping Variable: GRUPO

Figura D.3: Análisis de diferencias significativas en los datos de presentación al examen por parte de los participantes en la evaluación, entre los grupos GC y GT

**Ranks**

| | GRUPO | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| Presentado | GC | 56 | 61,42 | 3439,50 |
| | GV | 57 | 52,66 | 3001,50 |
| | Total | 113 | | |

**Test Statistics[a]**

| | Presentado |
|---|---|
| Mann-Whitney U | 1348,500 |
| Wilcoxon W | 3001,500 |
| Z | -1,924 |
| Asymp. Sig. (2-tailed) | ,054 |

a. Grouping Variable: GRUPO

Figura D.4: Análisis de diferencias significativas en los datos de presentación al examen por parte de los participantes en la evaluación, entre los grupos GC y GV

**Ranks**

| | GRUPO | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| Presentado | GV | 57 | 56,97 | 3247,50 |
| | GT | 54 | 54,97 | 2968,50 |
| | Total | 111 | | |

**Test Statistics[a]**

| | Presentado |
|---|---|
| Mann-Whitney U | 1483,500 |
| Wilcoxon W | 2968,500 |
| Z | -,401 |
| Asymp. Sig. (2-tailed) | ,688 |

a. Grouping Variable: GRUPO

Figura D.5: Análisis de diferencias significativas en los datos de presentación al examen por parte de los participantes en la evaluación, entre los grupos GV y GT

### D.3.2. Eficacia pedagógica

**One-Sample Kolmogorov-Smirnov Test**

| | | Práctica | Teoría | Problema | Total |
|---|---|---|---|---|---|
| N | | 116 | 116 | 116 | 116 |
| Normal Parameters[a,b] | Mean | ,5789 | ,6565 | ,3507 | ,4842 |
| | Std. Deviation | ,36926 | ,41439 | ,30716 | ,27094 |
| Most Extreme Differences | Absolute | ,247 | ,305 | ,134 | ,067 |
| | Positive | ,189 | ,218 | ,134 | ,067 |
| | Negative | -,247 | -,305 | -,127 | -,066 |
| Kolmogorov-Smirnov Z | | 2,665 | 3,286 | 1,438 | ,723 |
| Asymp. Sig. (2-tailed) | | ,000 | ,000 | ,032 | ,673 |

a. Test distribution is Normal.

b. Calculated from data.

Figura D.6: Test de normalidad para los datos cuantitativos de las calificaciones

**Ranks**

| | GRUPO | N | Mean Rank |
|---|---|---|---|
| Práctica | GC | 37 | 56,54 |
| | GV | 43 | 63,64 |
| | GT | 36 | 54,38 |
| | Total | 116 | |
| Teoría | GC | 37 | 62,03 |
| | GV | 43 | 64,06 |
| | GT | 36 | 48,24 |
| | Total | 116 | |
| Problema | GC | 37 | 64,59 |
| | GV | 43 | 60,63 |
| | GT | 36 | 49,69 |
| | Total | 116 | |

**Test Statistics[a,b]**

| | Práctica | Teoría | Problema |
|---|---|---|---|
| Chi-Square | 1,775 | 5,743 | 3,924 |
| df | 2 | 2 | 2 |
| Asymp. Sig. | ,412 | ,057 | ,141 |

a. Kruskal Wallis Test

b. Grouping Variable: GRUPO

Figura D.7: Test de dependencia entre las calificaciones cuantitativas de la práctica obligatoria (O), la cuestión teórica (C) y el problema (P); con los grupos de estudiantes

**ANOVA**

Total

| | Sum of Squares | df | Mean Square | F | Sig. |
|---|---|---|---|---|---|
| Between Groups | ,362 | 2 | ,181 | 2,528 | ,084 |
| Within Groups | 8,080 | 113 | ,072 | | |
| Total | 8,442 | 115 | | | |

Figura D.8: Test de dependencia ANOVA entre la calificación cuantitativa total (T) y los grupos de estudiantes

**Ranks**

| | GRUPO | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| C | GC | 37 | 39,68 | 1468,00 |
| | GV | 43 | 41,21 | 1772,00 |
| | Total | 80 | | |

**Test Statistics[a]**

| | C |
|---|---|
| Mann-Whitney U | 765,000 |
| Wilcoxon W | 1468,000 |
| Z | -,328 |
| Asymp. Sig. (2-tailed) | ,743 |

**Ranks**

| | GRUPO | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| C | GC | 37 | 41,35 | 1530,00 |
| | GT | 36 | 32,53 | 1171,00 |
| | Total | 73 | | |

**Test Statistics[a]**

| | C |
|---|---|
| Mann-Whitney U | 505,000 |
| Wilcoxon W | 1171,000 |
| Z | -1,879 |
| Asymp. Sig. (2-tailed) | ,060 |

**Ranks**

| | GRUPO | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| C | GV | 43 | 44,85 | 1928,50 |
| | GT | 36 | 34,21 | 1231,50 |
| | Total | 79 | | |

**Test Statistics[a]**

| | C |
|---|---|
| Mann-Whitney U | 565,500 |
| Wilcoxon W | 1231,500 |
| Z | -2,201 |
| Asymp. Sig. (2-tailed) | ,028 |

a. Grouping Variable: GRUPO

Figura D.9: Tests de diferencias significativas entre los tres grupos por parejas, para la cuestión teórica

**Ranks**

| | GRUPO | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| P | GC | 37 | 41,91 | 1550,50 |
| | GV | 43 | 39,29 | 1689,50 |
| | Total | 80 | | |

**Test Statistics[a]**

| | P |
|---|---|
| Mann-Whitney U | 743,500 |
| Wilcoxon W | 1689,500 |
| Z | -,504 |
| Asymp. Sig. (2-tailed) | ,615 |

**Ranks**

| | GRUPO | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| P | GC | 37 | 41,69 | 1542,50 |
| | GT | 36 | 32,18 | 1158,50 |
| | Total | 73 | | |

**Test Statistics[a]**

| | P |
|---|---|
| Mann-Whitney U | 492,500 |
| Wilcoxon W | 1158,500 |
| Z | -1,942 |
| Asymp. Sig. (2-tailed) | ,052 |

**Ranks**

| | GRUPO | N | Mean Rank | Sum of Ranks |
|---|---|---|---|---|
| P | GV | 43 | 43,34 | 1863,50 |
| | GT | 36 | 36,01 | 1296,50 |
| | Total | 79 | | |

**Test Statistics[a]**

| | P |
|---|---|
| Mann-Whitney U | 630,500 |
| Wilcoxon W | 1296,500 |
| Z | -1,430 |
| Asymp. Sig. (2-tailed) | ,153 |

a. Grouping Variable: GRUPO

Figura D.10: Tests de diferencias significativas entre los tres grupos por parejas, para el problema

**Group Statistics**

| | GRUPO | N | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|
| T | GC | 37 | ,5129 | ,24454 | ,04020 |
| | GT | 36 | ,4015 | ,27966 | ,04661 |

**Independent Samples Test**

| | | Levene's Test for Equality of Variances | |
|---|---|---|---|
| | | F | Sig. |
| T | Equal variances assumed | 1,914 | ,171 |
| | Equal variances not assumed | | |

| | | t-test for Equality of Means | | | |
|---|---|---|---|---|---|
| | | t | df | Sig. (2-tailed) | Mean Difference |
| T | Equal variances assumed | 1,813 | 71 | ,074 | ,11138 |
| | Equal variances not assumed | 1,809 | 69,208 | ,075 | ,11138 |

Figura D.11: Tests de diferencias significativas entre los grupos GC y GT, para la calificación total

**Group Statistics**

| | GRUPO | N | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|
| T | GC | 37 | ,5129 | ,24454 | ,04020 |
| | GV | 43 | ,5287 | ,27560 | ,04203 |

**Independent Samples Test**

| | | Levene's Test for Equality of Variances | |
|---|---|---|---|
| | | F | Sig. |
| T | Equal variances assumed | ,405 | ,526 |
| | Equal variances not assumed | | |

| | | t-test for Equality of Means | | | |
|---|---|---|---|---|---|
| | | t | df | Sig. (2-tailed) | Mean Difference |
| T | Equal variances assumed | -,268 | 78 | ,789 | -,01576 |
| | Equal variances not assumed | -,271 | 77,917 | ,787 | -,01576 |

Figura D.12: Tests de diferencias significativas entre los grupos GC y GV, para la calificación total

**Group Statistics**

| | GRUPO | N | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|
| T | GV | 43 | ,5287 | ,27560 | ,04203 |
| | GT | 36 | ,4015 | ,27966 | ,04661 |

**Independent Samples Test**

| | | Levene's Test for Equality of Variances | |
|---|---|---|---|
| | | F | Sig. |
| T | Equal variances assumed | ,423 | ,517 |
| | Equal variances not assumed | | |

| | | t-test for Equality of Means | | | |
|---|---|---|---|---|---|
| | | t | df | Sig. (2-tailed) | Mean Difference |
| T | Equal variances assumed | 2,028 | 77 | ,046 | ,12714 |
| | Equal variances not assumed | 2,026 | 74,185 | ,046 | ,12714 |

Figura D.13: Tests de diferencias significativas entre los grupos GV y GT, para la calificación total

**Resumen del procesamiento de los casos**

| | Casos | | | | | |
|---|---|---|---|---|---|---|
| | Válidos | | Perdidos | | Total | |
| | N | Porcentaje | N | Porcentaje | N | Porcentaje |
| GRUPO * CalifCualitativa | 116 | 100,0% | 0 | ,0% | 116 | 100,0% |

**Tabla de contingencia GRUPO * CalifCualitativa**

| | | | CalifCualitativa | | | | |
|---|---|---|---|---|---|---|---|
| | | | SS | AP | NT | SB | Total |
| GRUPO | GC | Recuento | 16 | 13 | 8 | 0 | 37 |
| | | % de GRUPO | 43,2% | 35,1% | 21,6% | ,0% | 100,0% |
| | GV | Recuento | 20 | 11 | 6 | 6 | 43 |
| | | % de GRUPO | 46,5% | 25,6% | 14,0% | 14,0% | 100,0% |
| | GT | Recuento | 22 | 8 | 6 | 0 | 36 |
| | | % de GRUPO | 61,1% | 22,2% | 16,7% | ,0% | 100,0% |
| Total | | Recuento | 58 | 32 | 20 | 6 | 116 |
| | | % de GRUPO | 50,0% | 27,6% | 17,2% | 5,2% | 100,0% |

**Pruebas de chi-cuadrado**

| | Valor | gl | Sig. asintótica (bilateral) |
|---|---|---|---|
| Chi-cuadrado de Pearson | 13,408 | 6 | ,037 |
| N de casos válidos | 116 | | |

Figura D.14: Tests de dependencias entre calificaciones cualitativas y grupos de estudiantes

### D.3.3. Correlación entre las calificaciones y los profesores

**Tabla de contingencia**

| | | | PRÁCTICA OBLIGATORIA (O) | | | | |
|---|---|---|---|---|---|---|---|
| | | | SS | AP | NT | SB | Total |
| PROFESOR | A | Recuento | 26 | 4 | 6 | 20 | 56 |
| | | % de PROFESOR | 46,4% | 7,1% | 10,7% | 35,7% | 100,0% |
| | | % de PRÁCTICA OBLIGATORIA (O) | 31,3% | 40,0% | 33,3% | 37,0% | 33,9% |
| | B | Recuento | 57 | 6 | 12 | 34 | 109 |
| | | % de PROFESOR | 52,3% | 5,5% | 11,0% | 31,2% | 100,0% |
| | | % de PRÁCTICA OBLIGATORIA (O) | 68,7% | 60,0% | 66,7% | 63,0% | 66,1% |
| Total | | Recuento | 83 | 10 | 18 | 54 | 165 |
| | | % de PROFESOR | 50,3% | 6,1% | 10,9% | 32,7% | 100,0% |
| | | % de PRÁCTICA OBLIGATORIA (O) | 100,0% | 100,0% | 100,0% | 100,0% | 100,0% |

**Pruebas de chi-cuadrado**

| | Valor | gl | Sig. asintótica (bilateral) |
|---|---|---|---|
| Chi-cuadrado de Pearson | ,651[a] | 3 | ,885 |
| Razón de verosimilitudes | ,647 | 3 | ,886 |
| Asociación lineal por lineal | ,424 | 1 | ,515 |
| N de casos válidos | 165 | | |

a. 1 casillas (12,5%) tienen una frecuencia esperada inferior a 5. La frecuencia mínima esperada es 3,39.

**Tabla de contingencia**

| | | | CUESTIÓN TEÓRICA (C) | | | | |
|---|---|---|---|---|---|---|---|
| | | | SS | AP | NT | SB | Total |
| PROFESOR | A | Recuento | 17 | 0 | 2 | 37 | 56 |
| | | % de PROFESOR | 30,4% | ,0% | 3,6% | 66,1% | 100,0% |
| | | % de CUESTIÓN TEÓRICA (C) | 33,3% | ,0% | 18,2% | 37,0% | 33,9% |
| | B | Recuento | 34 | 3 | 9 | 63 | 109 |
| | | % de PROFESOR | 31,2% | 2,8% | 8,3% | 57,8% | 100,0% |
| | | % de CUESTIÓN TEÓRICA (C) | 66,7% | 100,0% | 81,8% | 63,0% | 66,1% |
| Total | | Recuento | 51 | 3 | 11 | 100 | 165 |
| | | % de PROFESOR | 30,9% | 1,8% | 6,7% | 60,6% | 100,0% |
| | | % de CUESTIÓN TEÓRICA (C) | 100,0% | 100,0% | 100,0% | 100,0% | 100,0% |

**Pruebas de chi-cuadrado**

| | Valor | gl | Sig. asintótica (bilateral) |
|---|---|---|---|
| Chi-cuadrado de Pearson | 3,186[a] | 3 | ,364 |
| Razón de verosimilitudes | 4,262 | 3 | ,235 |
| Asociación lineal por lineal | ,319 | 1 | ,572 |
| N de casos válidos | 165 | | |

a. 3 casillas (37,5%) tienen una frecuencia esperada inferior a 5. La frecuencia mínima esperada es 1,02.

Figura D.15: Test de correlación de las calificaciones en la práctica obligatoria y en la cuestión teórica, con los profesores

**Tabla de contingencia**

| | | | PROBLEMA (P) | | | | |
|---|---|---|---|---|---|---|---|
| | | | SS | AP | NT | SB | Total |
| PROFESOR | A | Recuento | 38 | 10 | 4 | 4 | 56 |
| | | % de PROFESOR | 67,9% | 17,9% | 7,1% | 7,1% | 100,0% |
| | | % de PROBLEMA (P) | 35,5% | 31,3% | 28,6% | 33,3% | 33,9% |
| | B | Recuento | 69 | 22 | 10 | 8 | 109 |
| | | % de PROFESOR | 63,3% | 20,2% | 9,2% | 7,3% | 100,0% |
| | | % de PROBLEMA (P) | 64,5% | 68,8% | 71,4% | 66,7% | 66,1% |
| Total | | Recuento | 107 | 32 | 14 | 12 | 165 |
| | | % de PROFESOR | 64,8% | 19,4% | 8,5% | 7,3% | 100,0% |
| | | % de PROBLEMA (P) | 100,0% | 100,0% | 100,0% | 100,0% | 100,0% |

**Pruebas de chi-cuadrado**

| | Valor | gl | Sig. asintótica (bilateral) |
|---|---|---|---|
| Chi-cuadrado de Pearson | ,403[a] | 3 | ,940 |
| Razón de verosimilitudes | ,409 | 3 | ,938 |
| Asociación lineal por lineal | ,211 | 1 | ,646 |
| N de casos válidos | 165 | | |

a. 2 casillas (25,0%) tienen una frecuencia esperada inferior a 5. La frecuencia mínima esperada es 4,07.

**Tabla de contingencia**

| | | | CALIFICACIÓN TOTAL (T) | | | | |
|---|---|---|---|---|---|---|---|
| | | | SS | AP | NT | SB | Total |
| PROFESOR | A | Recuento | 18 | 15 | 21 | 2 | 56 |
| | | % de PROFESOR | 32,1% | 26,8% | 37,5% | 3,6% | 100,0% |
| | | % de CALIFICACIÓN TOTAL (T) | 31,6% | 30,0% | 44,7% | 18,2% | 33,9% |
| | B | Recuento | 39 | 35 | 26 | 9 | 109 |
| | | % de PROFESOR | 35,8% | 32,1% | 23,9% | 8,3% | 100,0% |
| | | % de CALIFICACIÓN TOTAL (T) | 68,4% | 70,0% | 55,3% | 81,8% | 66,1% |
| Total | | Recuento | 57 | 50 | 47 | 11 | 165 |
| | | % de PROFESOR | 34,5% | 30,3% | 28,5% | 6,7% | 100,0% |
| | | % de CALIFICACIÓN TOTAL (T) | 100,0% | 100,0% | 100,0% | 100,0% | 100,0% |

**Pruebas de chi-cuadrado**

| | Valor | gl | Sig. asintótica (bilateral) |
|---|---|---|---|
| Chi-cuadrado de Pearson | 4,125[a] | 3 | ,248 |
| Razón de verosimilitudes | 4,172 | 3 | ,244 |
| Asociación lineal por lineal | ,258 | 1 | ,611 |
| N de casos válidos | 165 | | |

a. 1 casillas (12,5%) tienen una frecuencia esperada inferior a 5. La frecuencia mínima esperada es 3,73.

Figura D.16: Test de correlación de las calificaciones en el problema y en la calificación total, con los profesores

### D.3.4. Correlación entre la asignatura evaluada y Cálculo como asignatura de control

**Resumen del procesamiento de los casos**

| | Casos | | | | | |
|---|---|---|---|---|---|---|
| | Válidos | | Perdidos | | Total | |
| | N | Porcentaje | N | Porcentaje | N | Porcentaje |
| CALIFICACIÓN BLP * CALIFICACIÓN CÁLCULO * GRUPO | 94 | 75,2% | 31 | 24,8% | 125 | 100,0% |

**Tabla de contingencia CALIFICACIÓN BLP * CALIFICACIÓN CÁLCULO * GRUPO**

Recuento

| GRUPO | | | CALIFICACIÓN CÁLCULO | | | | | Total |
|---|---|---|---|---|---|---|---|---|
| | | | NP | SS | AP | NT | SB | |
| GC | CALIFICACIÓN BLP | NP | 2 | 3 | 1 | | 0 | 6 |
| | | SS | 3 | 2 | 1 | | 0 | 6 |
| | | AP | 2 | 1 | 3 | | 1 | 7 |
| | | NT | 3 | 0 | 1 | | 1 | 5 |
| | Total | | 10 | 6 | 6 | | 2 | 24 |
| GV | CALIFICACIÓN BLP | NP | 4 | 1 | 1 | 0 | 0 | 6 |
| | | SS | 1 | 5 | 4 | 0 | 0 | 10 |
| | | AP | 4 | 4 | 1 | 0 | 0 | 9 |
| | | NT | 0 | 2 | 1 | 0 | 1 | 4 |
| | | SB | 0 | 3 | 0 | 1 | 1 | 5 |
| | Total | | 9 | 15 | 7 | 1 | 2 | 34 |
| GT | CALIFICACIÓN BLP | NP | 6 | 3 | 2 | 0 | | 11 |
| | | SS | 5 | 3 | 2 | 0 | | 10 |
| | | AP | 5 | 3 | 0 | 0 | | 8 |
| | | NT | 2 | 1 | 1 | 1 | | 5 |
| | | SB | 0 | 1 | 1 | 0 | | 2 |
| | Total | | 18 | 11 | 6 | 1 | | 36 |

**Medidas simétricas**

| GRUPO | | | Valor | Error típ. asint.[a] | T aproximada[b] | Sig. aproximada |
|---|---|---|---|---|---|---|
| GC | Ordinal por ordinal | Tau-b de Kendall | ,096 | ,183 | ,523 | ,601 |
| | N de casos válidos | | 24 | | | |
| GV | Ordinal por ordinal | Tau-b de Kendall | ,255 | ,142 | 1,745 | ,081 |
| | N de casos válidos | | 34 | | | |
| GT | Ordinal por ordinal | Tau-b de Kendall | ,123 | ,149 | ,819 | ,413 |
| | N de casos válidos | | 36 | | | |

a. Asumiendo la hipótesis alternativa.

b. Empleando el error típico asintótico basado en la hipótesis nula.

Figura D.17: Test de correlación de las calificaciones obtenidas en la asignatura evaluada con respecto a Cálculo

**Resumen del procesamiento de los casos**

| | Casos | | | | | |
|---|---|---|---|---|---|---|
| | Válidos | | Perdidos | | Total | |
| | N | Porcentaje | N | Porcentaje | N | Porcentaje |
| NP/AP/SS BLP * NPT/AP/SS CÁLCULO * GRUPO | 94 | 75,2% | 31 | 24,8% | 125 | 100,0% |

**Tabla de contingencia NP/AP/SS BLP * NPT/AP/SS CÁLCULO * GRUPO**

Recuento

| GRUPO | | | NPT/AP/SS CÁLCULO | | | Total |
|---|---|---|---|---|---|---|
| | | | NP | SS | AP | |
| GC | NP/AP/SS BLP | NP | 2 | 3 | 1 | 6 |
| | | SS | 3 | 2 | 1 | 6 |
| | | AP | 5 | 1 | 6 | 12 |
| | Total | | 10 | 6 | 8 | 24 |
| GV | NP/AP/SS BLP | NP | 4 | 1 | 1 | 6 |
| | | SS | 1 | 5 | 4 | 10 |
| | | AP | 4 | 9 | 5 | 18 |
| | Total | | 9 | 15 | 10 | 34 |
| GT | NP/AP/SS BLP | NP | 6 | 3 | 2 | 11 |
| | | SS | 5 | 3 | 2 | 10 |
| | | AP | 7 | 5 | 3 | 15 |
| | Total | | 18 | 11 | 7 | 36 |

**Medidas simétricas**

| GRUPO | | | Valor | Error típ. asint.[a] | T aproximada[b] | Sig. aproximada |
|---|---|---|---|---|---|---|
| GC | Ordinal por ordinal | Tau-b de Kendall | ,125 | ,182 | ,685 | ,493 |
| | N de casos válidos | | 24 | | | |
| GV | Ordinal por ordinal | Tau-b de Kendall | ,122 | ,170 | ,711 | ,477 |
| | N de casos válidos | | 34 | | | |
| GT | Ordinal por ordinal | Tau-b de Kendall | ,051 | ,149 | ,341 | ,733 |
| | N de casos válidos | | 36 | | | |

a. Asumiendo la hipótesis alternativa.

b. Empleando el error típico asintótico basado en la hipótesis nula.

Figura D.18: Test de correlación de no pesentados/aprobados/suspensos obtenidas en la asignatura evaluada con respecto a Cálculo

# Bibliografía


[1] T. Ahoniemi and E. Lahtinen. Visualizations in preparing for programming exercise sessions. *Electronic Notes in Theoretical Computer Science*, 178:137–144, 2007.

[2] A. Akingbade, T. Finley, D. Jackson, P. Patel, and S.H. Rodger. JAWAA: easy web-based animation from CS 0 to advanced CS courses. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pages 162–166, New York, NY, USA, 2003. ACM Press.

[3] E. Albert, M. Hanus, F. Huch, J. Oliver, and G. Vidal. An operational semantics for declarative multi-paradigm languages. In *Proceedings of the 11th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2002)*, pages 7–20, Grado (Italy), 2002. Research Report UDMI/18/2002/RR, Università degli Studi di Udine.

[4] J.M. Anderson and T.L. Naps. A context for the assessment of algorithm visualization system as pedagogical tools. In *Proceedings of the First Program Visualization Workshop*, International Proceedings Series, pages 121–130, Joensuu, Finland, 2001. University of Joensuu.

[5] E. Angel and D. Morrison. Speeding up bresenham's algorithm. *IEEE Computer Graphics and Applications*, 11:16–17, 1991.

[6] M. Auguston and J. Reinfelds. A visual miranda machine. In *Proceedings of the Software Education Conference (SRIT-ET'94)*, pages 198–203. IEEE Computer Society Press, 1994.

[7] R.M. Baecker. *Interactive Computer-Mediated Animation*. PhD thesis, M.I.T. Department of Electrical Engineering, April 1969.

[8] R.M. Baecker. Sorting out sorting. SIGGRAPH Video Review 7, 1983.

[9] R.M. Baecker. Sorting out sorting: A case study of software visualization for teaching computer science. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization. Programming as a Multimedia Experience*, pages 369–381. MIT Press, 1998.

[10] R.M. Baecker and A. Marcus. Printing and publishing C programs. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization. Programming as a Multimedia Experience*, pages 45–61. MIT Press, 1998.

[11] R.M. Baecker and B.A. Price. The early history of software visualization. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization. Programming as a Multimedia Experience*, pages 29–34. MIT Press, 1998.

[12] L. Bartram, A. Ho, J. Dill, and F. Henigman. The continuous zoom: A constrained fisheye technique for viewing and navigating large information spaces. In *Proceedings of the 8th annual ACM symposium on User Interface Software and Technology (UIST'95)*, pages 207–215, New York, NY, USA, 1995. ACM Press.

[13] G.A. Baxes. *Digital image processing principles and applications*. John Wiley & Sons, 1994.

[14] J. Bazik, R. Tamassia, S. P. Reiss, and A. van Dam. Software visualization in teaching at brown university. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization. Programming as a Multimedia Experience*, pages 382–398. MIT Press, Cambridge, MA, 1998.

[15] H. Böcker, G. Fischer, and H. Nieper. The enhancement of understanding throuhg visual representations. In *CHI '86: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 44–50, New York, NY, USA, 1986. ACM Press.

[16] D.B. Beard and J.Q. Walker. Navigational techniques to improve the display of large two-dimensional spaces. *Behaviour and Information Technology*, 9(6):451–466, 1990.

[17] B.B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering*, 30(8):535–546, 2004.

[18] B.B. Bederson and J.D. Hollan. Pad++: a zooming graphical interface for exploring alternate interface physics. In *Proceedings of the 7th annual ACM symposium on User Interface Software and Technology (UIST'94)*, pages 17–26, New York, NY, USA, 1994. ACM Press.

[19] A. Beguelin, J. Dongarra, A. Geist, and V. Sunderman. Visualisation and debugging in a heterogeneous environment. *IEEE Computer*, 26(6):88–95, June 1993.

[20] M. Ben-Ari. Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1):45–73, 2001.

[21] M. Ben-Ari. Program visualization in theory and practice. *Informatik/Informatique*, 2:8–11, April 2001.

[22] R. Ben-Bassat, M. Ben-Ari, and P.A. Uronen. The jeliot 2000 program animation system. *Computers & Education*, 40(1):1–15, January 2003.

[23] R. Berghammer and U. Milanese. Kiel - a computer system for visualizing the execution of functional programs. In M. Hanus, editor, *Functional and (Constraint) Logic Programming, WFLP 2001*, pages 365–368, Kiel, Germany, 2001. Christian-Albrechts-Universitt zu Kiel. Report No. 2017.

[24] J. Bertin. *Graphics and Graphic Information Processing*. Walter de Gruyter, 1981.

[25] S. Björk. *Flip Zooming. The Development of an Information Visualization Technique*. PhD thesis, Göteborg University, Dept. of Informatics, 2000.

[26] S. Björk, L.E Holmquist, and J. Redström. A framework for focus+context visualization. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'99)*, pages 53–56, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[27] S. Björk and J. Redström. Redefining the focus and context of focus+context visualizations. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'00)*, pages 85–90, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[28] B. Bloom, E. Furst, W. Hill, and D.R. Krathwohl. *Taxonomy of Educational Objetives: Handbook I, The Cognitive Domain*. Addison-Wesley, 1959.

[29] V. Bonifaci, C. Demetrescu, I. Finocchi, and L. Laura. Visual editing of animated algorithms: the Leonardo Web builder. In *AVI '06: Proceedings of the working Conference on Advanced Visual Interfaces*, pages 476–479, New York, NY, USA, 2006. ACM Press.

[30] B. Bra$\beta$el, O. Chitil, M. Hanus, and F. Huch. Observing functional logic computations. In *Proceedings of the Sixth International Symposium on Practical Aspects of Declarative Languages, PADL'04*, volume 3057 of *Lecture Notes in Computer Science*, pages 193–208, Berlin, Germany, 2004. Springer-Verlag.

[31] J.E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[32] J.E. Bresenham. Incremental line compaction. *The Computer Journal*, 25(1):116–120, 1982.

[33] M.H. Brown. Perspectives on algorithm animation. In *CHI '88: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 33–38, New York, NY, USA, 1988. ACM Press.

[34] M.H. Brown. ZEUS: A system for algorithm animation and multi-view editing. In *Proceedings of the IEEE Workshop on Visual Languages*, pages 4–9, 1991.

[35] M.H. Brown. A taxonomy of algorithm animation displays. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization. Programming as a Multimedia Experience*, pages 35–42. MIT Press, 1998.

[36] M.H. Brown and R. Sedgewick. A system for algorithm animation. In *Proceedings of the ACM SIGGRAPH'84*, pages 177–186, New York, NY, USA, 1984. ACM Press.

[37] M. Bruce-Lockhart, T.S. Norvell, and Y. Cotronis. Program and algorithm visualization in engineering and physics. *Electronic Notes in Theoretical Computer Science*, 178:111–119, 2007.

[38] P. Brusilovsky and T.D. Loboda. WADEIn II: a case for adaptive explanatory visualization. In *ITiCSE '06: Proceedings of the 11th annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 48–52, New York, NY, USA, 2006. ACM Press.

[39] M.D. Byrne, R. Catrambone, and J.T. Stasko. Do algorithm animations aid learning? Technical Report GIT-GVU-96-18, Georgia Institute of Technology, Atlanta, GA, USA, 1996.

[40] A.E.R. Campbell, G.L. Catto, and E.E. Hansen. Language-independent interactive data visualization. In *SIGCSE '03: Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pages 215–219, New York, NY, USA, 2003. ACM Press.

[41] S.K. Card and J.D. Mackinlay. The structure of the information visualization design space. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'97)*, pages 92–99, Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[42] S.K. Card, J.D. Mackinlay, and B. Shneidermann, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, 1999.

[43] K.R. Castleman. *Digital image processing*. Prentice Hall, 1996.

[44] C. Chen, editor. *Information Visualization. Beyond the Horizon*. Springer-Verlag, 2004.

[45] J.X. Chen. Multiple segment line scan-conversion. *COMPUTER GRAPHICS forum*, 16(5):257–268, 1997.

[46] G. Cheng and N.A.B. Gray. A program visualization tool. Technical report, University of Woolongong, 1992.

[47] E. Chi. A taxonomy of visualization techniques using the data state reference model. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'00)*, pages 69–75, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[48] L. Cohen, L. Manion, and K. Morrison. *Research Methods in Education*. RoutledgeFalmer, 5th edition, 2005.

[49] World Wide Web Consortium. Cascading style sheets. http://www.w3.org/Style/CSS/, 2007.

[50] World Wide Web Consortium. Extensible markup language (xml). http://www.w3.org/XML/, 2007.

[51] World Wide Web Consortium. Frames in html documents. inline frames: the iframe element. http://www.w3.org/TR/html4/present/frames.html#h-16.5, 2007.

[52] World Wide Web Consortium. W3c html. http://www.w3.org/html/, 2007.

[53] World Wide Web Consortium. Xhtml(tm) 2.0. http://www.w3.org/TR/xhtml2/, 2007.

[54] World Wide Web Consortium. Xsl transformations (xslt). http://www.w3.org/TR/xslt, 2007.

[55] P. Crescenzi, C. Demetrescu, I. Finocchi, and R. Petreschi. Reversible execution and visualization of programs with leonardo. *Journal of Visual Languages and Computing*, 11(2):125–150, 2000.

[56] P. Crescenzi and C. Nocentini. Fully integrating algorithm visualization into a cs2 course.: a two-year experience. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 296–300, New York, NY, USA, 2007. ACM Press.

[57] J.H. Cross, T.D. Hendrix, J. Jain, and L.A. Barowski. Dynamic object viewers for data structures. In *SIGCSE '07: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, pages 4–8, New York, NY, USA, 2007. ACM Press.

[58] C. Demetrescu and I. Finocchi. A general-purpose logic-based visualization framework. In *Proceedings of the 7th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media (WSCG'99)*, pages 55–62, 1999.

[59] H.L. Dershem, R.L. McFall, and N. Uti. Animation of java linked lists. In *SIGCSE '02: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, pages 53–57, New York, NY, USA, 2002. ACM Press.

[60] S. Diehl, editor. *Software Visualisation*, volume 2269 of *Lecture Notes in Computer Science State-of-the-Art Survey*. Springer-Verlag, 2002.

[61] S. Diehl and A. Kerren. Levels of exploration. In *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pages 60–64, New York, NY, USA, 2001. ACM Press.

[62] J. Domingue, B.A. Price, and M. Eisenstadt. A framework for describing and implementing software visualization systems. In *Proceedings of Graphics Interface '92*, pages 53–60, 1992.

[63] P. Eades and K. Zhang. *Software Visualisation*. World Scientific Publishing Co, 1996.

[64] M. Eisenstadt and M. Brayshaw. The transparent prolog machine (TPM): An execution model and graphical debugger for logic programming. *Journal of logic programming*, 5(4):277–342, 1988.

[65] M. Esponda-Argüero and R. Rojas. Learning algorithms with an electronic chalkboard over the web. In *Advances in Web-Based Learning - ICWL 2004*, volume 3143 of *Lecture Notes in Computer Science*, pages 1–10, Berlin Heidelberg New York, 2004. Springer-Verlag.

[66] D. Field. Incremental linear interpolation. *ACM Transactions on Graphics*, 4(1):1–11, January 1985.

[67] R.B. Findler. DrScheme: A programming environment for Scheme. *Journal of Functional Programming*, 12(2):159–182, March 2002.

[68] S.P. Foubister and C. Runciman. Techniques for simplifying the visualization of graph reduction. In K. Hammond, D.N. Turner, and P.M. Sansom, editors, *Functional Programming*, pages 65–77. Springer-Verlag, Berlin, Germany, 1995.

[69] G.W. Furnas. Generalized fisheye views. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '86)*, pages 16–23, New York, NY, USA, 1986. ACM Press.

[70] G.W. Furnas. The fisheye view: A new look at structured files. In S.K. Card, J.D. Mackinlay, and B. Shneiderman, editors, *Information Visualization: Using Vision to Think*, pages 145–152. Morgan Kaufmann Publishers, 1999.

[71] D.L. Gabel, editor. *Handbook of research on Science Teaching and Learning*. Simon & Schuster Macmillan, New York, NY, US, 1994.

[72] P.L. Gardner. Modifications of bresenham's algorithms for display. *IBM Tech. Disclosure Bull.*, 18:1595–1596, 1975.

[73] P. Gestwicki and B. Jayaraman. Methodology and architecture of JIVE. In *SoftVis '05: Proceedings of the 2005 ACM Symposium on Software Visualization*, pages 95–104, New York, NY, USA, 2005. ACM Press.

[74] A. Gill. Debugging Haskell by observing intermediate data structures. *Electronic Notes in Theoretical Computer Science. Proceedings of the 2000 ACM SIGPLAN Haskell Workshop*, 41(1), 2000.

[75] G.W. Gill. N-step incremental straight-line algorithms. In *IEEE Computer Graphics and Applications*, volume 14, pages 66–72, Labtam Australia Pty. Ltd., Mordialloc, Vic., 1994.

[76] P.A. Gloor. Animated algorithms. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization. Programming as a Multimedia Experience*, pages 409–416. MIT Press, 1998.

[77] F. Gortazar-Bellas, J. Urquiza-Fuentes, and J.Á. Velázquez-Iturbide. Reduction and flip-zooming in comprenhension-preserving miniatures of program visualizations. In *Proceedings of the Third IASTED Symposium on Visualization, Imaging and Image Processing VIIP 2003*, pages 855–861, Zurich, Switzerland, 2003. ACTA Press.

[78] S. Grissom, M.F. McNally, and T.L. Naps. Algorithm visualization in CS education: comparing levels of student engagement. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, pages 87–94, New York, NY, USA, 2003. ACM Press.

[79] F. Hadlock, J. Williams, J. Wyatt, D. Balasubramanian, J. Bittinger, C. Davis, S. Kesiraju, J. Kolpack, J. Northcutt, S. Sudireddy, and M. Tyler. An internet based algorithm visualization system. *Journal of Computing Sciences in Colleges*, 20(2):304–310, december 2004.

[80] J. Hamer. A lightweight visualizer for java. In *Proceedings of the Third Program Visualization Workshop*, pages 54–61, Coventry, UK, 2004. University of Warwick.

[81] J. Hamer. Visualising java data structures as graphs. In R. Lister and A. Young, editors, *ACE'04 Australasian Computer Society Education Conference, volume 30 of Conferences in Research and Practice in Information Technology*, pages 125–129. Australian Computer Society, 2004.

[82] A.G. Hamilton-Taylor and E. Kraemer. Ska: supporting algorithm and data structure discussion. In *SIGCSE '02: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, pages 58–62, New York, NY, USA, 2002. ACM Press.

[83] S.R. Hansen, N.H. Narayanan, and D. Schrimpsher. Helping learners visualize and comprehend algorithms. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, 2(1), April 2000. available at http://imej.wfu.edu/articles/2000/1/02/, 2007.

[84] M. Hanus and J. Koj. CIDER: An Integrated Development Environment for Curry. In M. Hanus, editor, *Proceedings of the International Workshop on Functional and (Constraint) Logic Programming, WFLP 2001*, pages 369–373, Kiel, Germany, September 2001. Christian-Albrechts-Universitt zu Kiel. Report No. 2017.

[85] M.T. Heath and J.A. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, 8(5):29–39, September 1991.

[86] T.D. Hendrix, J.H. Cross, and L.A. Barowski. An extensible framework for providing dynamic data structure visualizations in a lightweight ide. In *SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pages 387–391, New York, NY, USA, 2004. ACM Press.

[87] L.E. Holmquist. *Breaking the Screen Barrier*. PhD thesis, Göteborg University, Dept. of Informatics, 2000.

[88] L.E. Holmquist and S. Björk. A hierarchical focus + context method for image browsing. In *SIGGRAPH 98 Sketches and Applications*, page 282, New York, NY, USA, 1998. ACM Press.

[89] K. Hornbæk. *Usability of Information Visualization: Reading and Interaction Processes.* PhD thesis, University of Copenhagen. Department of Computing, 2001.

[90] T. Hübscher-Younger and N.H. Narayanan. Dancing hamsters and marble statues: characterizing student visualizations of algorithms. In *SoftVis '03: Proceedings of the 2003 ACM Symposium on Software Visualization*, pages 95–104, New York, NY, USA, 2003. ACM Press.

[91] C.D. Hundhausen. Integrating algorithm visualization technology into an undergraduate algorithms course: ethnographic studies of a social constructivist approach. *Computers & Education*, 39:237–260, 2002.

[92] C.D. Hundhausen and J.L. Brown. Personalizing and discussing algorithms within CS1 studio experiences: an observational study. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 45–56, New York, NY, USA, 2005. ACM Press.

[93] C.D. Hundhausen and J.L. Brown. What you see is what you code: A "radically-dynamic" algorithm visualization development model for novice learners. In *Proceedings IEEE 2005 Symposium on Visual Languages and Human-Centric Computing*, pages 163–170, Los Alamitos, CA, USA, 2005. IEEE Computer Society Press.

[94] C.D. Hundhausen, S.A. Douglas, and J.T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, 2002.

[95] T. Igarishi and K. Hinckley. Speed-dependent automatic zooming for browsing large documents. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST 2000)*, pages 139–148, New York, NY, USA, 2000. ACM Press.

[96] P. Ihantola, V. Karavirta, A. Korhonen, and J. Nikander. Taxonomy of effortless creation of algorithm visualizations. In *ICER '05: Proc. 2005 International Workshop on Computing Education Research*, pages 123–133, New York, NY, USA, 2005. ACM Press.

[97] ISO. Information processing – text and office systems – standard generalized markup language (sgml). ISO 8879:1986, 1986.

[98] ISO. Ergonomic requirements for office work with visual display terminals (vdts) - part 11: Guidance on usability. ISO 9241-11, 1998.

[99] D.J. Jarc and M.B. Feldman. An empirical study of web-based algorithm animation courseware in an ada data structure course. In *SIGAda '98: Proceedings of the 1998 annual ACM SIGAda international conference on Ada*, pages 68–74, New York, NY, USA, 1998. ACM Press.

[100] B. Jähne. *Digital Image Processing.* Springer-Verlag, Berlin, Germnay.

[101] B. Jähne, H. Haussecker, and P. Geissler, editors. *Handbook of computer vision and applications*, volume 2. Academinc Press, 1999.

[102] R. Jiménez-Peris, C. Pareja-Flores, M. Patiño-Martínez, and J.Á. Velázquez-Iturbide. Graphical visualization of the evaluation of functional programs. In *ITiCSE '96: Proceedings of the 1st conference on Integrating Technology into Computer Science Education*, pages 36–38, New York, NY, USA, 1996. ACM Press.

[103] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd conference on Visualization '91*, pages 284 – 291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[104] S.P. Jones, editor. *Haskell 98 language and libraries: the Revised Report*. Cambridge University Press, 2003.

[105] V. Karavirta. Integrating algorithm visualization systems. *Electronic Notes in Theoretical Computer Science*, 178:79–87, 2007.

[106] V. Karavirta, A. Korhonen, L. Malmi, and K. Stålnacke. Matrixpro - a tool for on-the-fly demonstration of data structures and algorithms. In *Proceedings of the Third Program Visualization Workshop*, pages 26–33, Coventry, UK, 2004. University of Warwick.

[107] V. Karavirta, A. Korhonen, and P. Tenhunen. Survey of effortlessness in algorithm visualization systems. In *Proceedings of the Third Program Visualization Workshop*, pages 141–148, Coventry, UK, 2004. University of Warwick.

[108] N. Kawaguchi, T. Sakabe, and Y. Inagaki. TERSE: TErm Rewriting Support Environment. In *Proceedings of the 1994 ACM SIGPLAN Workshop on Standrad ML and its Applications*, pages 91–100, 1994.

[109] C. Kehoe, J.T. Stasko, and A. Taylor. Rethinking the evaluation of algorithm animations as learning aids: An observational study. git-gvu-99-10. Technical report, Georgia Institute of Technology, 1999.

[110] C. Kehoe, J.T. Stasko, and A. Taylor. Rethinking the evaluation of algorithm animations as learning aids: An observational study. *International Journal of Human-Computer Studies*, 54(2):265–284, 2001.

[111] T. Kientzle. Scaling bitmaps with bresenham. *C/C++ User's Journal*, pages 51–53, October 1995.

[112] D. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[113] J.A. Kohl and G.A. Geist. The PVM 3.4 tracing facility and XPVM 1.1. In *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, volume 1, pages 290–299, 1996.

[114] I. Koifman, I. Shimshoni, and A. Tal. MAVIS: A multi-level algorithm visualization system within a collaborative distance learning environment. *Journal of Visual Languages and Computing*, 2006. In press. doi:10.1016/j.jvlc.2006.09.004.

[115] A. Korhonen and L. Malmi. Algorithm simulation with automatic assessment. In *Proceedings of the 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, pages 160–163, New York, NY, USA, 2000. ACM Press.

[116] A. Korhonen, L. Malmi, P. Silvasti, V. Karavirta, J. Lönnberg, J. Nikander, K. Stålnacke, and P. Ihantola. Matrix - a framework for interactive software visualization. Technical Report TKO-B 154/04, Laboratory of Information Processing Science, Department of Computer Science and Engineering, Helsinki University of Technology, Helsinki, Finland, 2004.

[117] A.N. Kumar. Results from the evaluation of the effectiveness of an online tutor on expression evaluation. In *SIGCSE '05: Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, pages 216–220, New York, NY, USA, 2005. ACM Press.

[118] M-J. Laakso, T. Salakoski, L. Grandell, X. Qiu, A. Korhonen, and L. Malmi. Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2. *Informatics in Education*, 4(1):49Ũ–68, 2005.

[119] P. LaFollette, J. Korsh, and R. Sangwan. A visual interface for effortless animation of c/c++ programs. *Journal of Visual Languages and Computing*, 11:27–48, 2000.

[120] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI '95: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[121] M.V. LaPolla. Towards a theory of abstraction and visualization for debugging massively parallel programs. In *Proceedings of the Hawaii International Conference on System Sciences*, volume 2, pages 184–195, 1992.

[122] J. Larkin and H.A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11:65–99, 1987.

[123] K. Lau, R.A. Rensink, and T. Munzner. Perceptual invariance of nonlinear focus+context transformations. In *APGV '04: Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization*, pages 65–72, New York, NY, USA, 2004. ACM Press.

[124] A.W. Lawrence, A.M. Badre, and J.T. Stasko. Empirically evaluating the use of animations to teach algorithms. In *IEEE Symposium on Visual Languages, 1994. Proceedings.*, pages 48–54, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[125] S.G. Lazzeri. A graphical enironment for monitoring prolog programs. Master's thesis, Department of Electrical Engineering and Computer Science, The George Washington University, Washington D.C., USA, 1991.

[126] T. Lehr, Z. Segall, D.F. Vrsalovic, E. Caplan, A.L. Chung, and C.E. Fineman. Visualisation perfonmance debugging. *IEEE Computer*, 22(10):38–51, October 1989.

[127] Y.K. Leung and M.D Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1(2):126–160, 1994.

[128] R.B. Levy and M. Ben-Ari. We work so hard and they don't use it: acceptance of software tools by teachers. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 246–250, New York, NY, USA, 2007. ACM Press.

[129] H. Lieberman and C. Fry. ZStep95: A reversible, animated source code stepper. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization. Programming as a Multimedia Experience*, pages 277–292. MIT Press, 1998.

[130] T.D. Loboda, A. Frengov, A.N. Kumar, and P. Brusilovsky. Distributed framework for adaptive explanatory visualization. *Electronic Notes in Theoretical Computer Science*, 178:145–152, 2007.

[131] J.D. Mackinlay, G.G. Robertson, and S.K. Card. The perspective wall: Detail and context smoothly integrated. In *Conference proceedings on Human Factors in Computing Systems '91 (CHI '91)*, pages 173–179, New York, NY, USA, November 1991. ACM Press.

[132] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: Trakla2. *Informatics in Education*, 3(2):267Ű–288, 2004.

[133] R.E. Mayer. Are we asking the right questions? *Educational Psychologist*, 32(1):1–19, 1997.

[134] R.E. Mayer and R.B. Anderson. Animations need narrations: an experimental test of a dual-coding hypothesis. *Journal of Educational Psychology*, 83:484–490, Feb. 1991.

[135] M.A. Medina-Sánchez, C.A. Lázaro-Carrascosa, C. Pareja-Flores, J. Urquiza-Fuentes, and J.Á. Velázquez-Iturbide. Empirical evaluation of usability of animations in a functional programming environment. Technical report, Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 2004. Technical report 141/04.

[136] Sun Microsystems. Scale area averaging. http://java.sun.com/j2se/1.5.0/docs/api/java/awt/image/AreaAveragingScaleFilter.html, 2007.

[137] Sun Microsystems. Scale replicate. http://java.sun.com/j2se/1.5.0/docs/api/java/awt/image/ReplicateScaleFilter.html, 2007.

[138] R. Milner, M. Tofte, and R. Harper. *The definition of Standard ML.* MIT Press, 1990.

[139] A. Moreno. Algorithm animation. In a. Kerren, A. Ebert, and J. Meyer, editors, *Human-Centered Visualization Environments*, volume 4417 of *Lecture Notes in Computer Science*, pages 297–311. Springer-Verlag, Berlin Heidelberg New York, 2007.

[140] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari. Visualizing programs with jeliot 3. In *AVI '04: Proceedings of the working Conference on Advanced Visual Interfaces*, pages 373–376, New York, NY, USA, 2004. ACM Press.

[141] B. Moskal, D. Lurie, and S. Cooper. Evaluating the effectiveness of a new instructional approach. In *Proceedings of the Technical Symposium on Computer Sceince Education, SIGCSE'04*, pages 75–79, Norfolk, Virginia, USA, March 2004.

[142] P. Mulholland and M. Eisenstadt. Using software to teach computer programming: Past present and future. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization. Programming as a Multimedia Experience*, pages 399–408. MIT Press, 1998.

[143] B.A. Myers. Visual programming, programming by example, and program visualization: A taxonomy. In *Proceedings of Human Factors in Computing Systems (CHI '86)*, pages 59–66, New York, NY, USA, 1886. ACM Press.

[144] N. Myller. Automatic generation of prediction questions during program visualization. *Electronic Notes in Theoretical Computer Science*, 178:43–49, 2007.

[145] F. Naharro-Berrocal, C. Pareja-Flores, J. Urquiza-Fuentes, and J.Á. Velázquez-Iturbide. Approaches to comprehension-preserving graphical reduction of program visualizations. In *Proceedings del ACM Symposium on Applied Computing 2002*, pages 771–777, New York, NY, USA, 2002. ACM Press.

[146] F. Naharro-Berrocal, C. Pareja-Flores, J. Urquiza-Fuentes, J.Á. Velázquez-Iturbide, and F. Gortazar-Bellas. Redesigning the animation capabilities of a functional programming environment under an educational framework. In M. Ben-Ari, editor, *DAIMI PB - 567 Proceedings of the Second Program Visualization Workshop*, pages 60–69, HornstrupCentret, Denmark, 2002. University of Aarhus, Department of Computer Science.

[147] F. Naharro-Berrocal, C. Pareja-Flores, J.Á. Velázquez-Iturbide, and Martínez-Santamarta. Automatic Web publishing of algorithm animations. *Informatik/Informatique*, 2:41–45, April 2001.

[148] F. Naharro-Berrocal, C. Pareja-Flores, and J.Á. Velázquez-Iturbide. Automatic generation of algorithm animations in a programming environment. In *Proceedings of the 2000 Frontiers in Education Conference*, pages 6–12 (session S2C), Champaign, IL, USA, 2000. Stipes Publishing.

[149] F. Naharro-Berrocal, C. Pareja-Flores, and J.Á. Velázquez-Iturbide. Foundations for the automatic construction of animations and their application to functional programs. In *Proceedings of the First Program Visualization Workshop*, International Proceedings Series, pages 29–40, Joensuu, Finland, 2001. University of Joensuu.

[150] F. Naharro-Berrocal, C. Pareja-Flores, and J.Á. Velázquez-Iturbide. Toward friendly, dialog-based construction of algorithm visualizations and animations. In *Proceedings of the IASTED International Conference Visualization, Imaging, and Image Processing*, pages 27–32, Zurich, Switzerland, 2001. ACTA Press.

[151] F. Naharro-Berrocal, A. Velázquez-Iturbide, C. Pareja-Flores, and M.A. Medina-Sánchez. Valoración de la eficacia de las animaciones de algoritmos en el aprendizaje de la programación funcional. Working paper, 2001.

[152] M. Najork. Web-based algorithm animation. In *DAC '01: Proceedings of the 38th Conference on Design Automation*, pages 506–511, New York, NY, USA, 2001. ACM Press.

[153] T.L. Naps. Algorithm visualization in computer science laboratories. In *SIGCSE '90: Proceedings of the twenty-first SIGCSE Technical Symposium on Computer Science Education*, pages 105–110, New York, NY, USA, 1990. ACM Press.

[154] T.L. Naps. JHAVE: supporting algorithm visualization. *IEEE Computer Graphics and Applications*, 25:49–55, Sept.-Oct. 2005.

[155] T.L. Naps, S. Cooper, B. Koldehofe, C. Leska, G. Rößling, W. Dann, A. Korhonen, L. Malmi, J. Rantakokko, R.J. Ross, J. Anderson, R. Fleischer, M. Kuittinen, and M. McNally. Iticse 2003 working group reports: Evaluating the educational impact of visualization. *ACM SIGCSE Bulletin, Working group reports from ITiCSE on Innovation and technology in computer science education*, 35(4):124–136, june 2003.

[156] T.L. Naps, J. Eagan, and L. Norton. JHAVÉ: An environment to actively engage students in web-based algorithm visualizations. In *31st SIGCSE Technical Symposium on Computer Science Education*, pages 109–113, New york, NY, USA, 2000. ACM Press.

[157] T.L. Naps and G. Rößling. JHAVÉ – more visualizers (and visualizations) needed. *Electronic Notes in Theoretical Computer Science*, 178:33–41, 2007.

[158] T.L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C.D. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S.H. Rodger, and J.Á. Velázquez-Iturbide. Iticse 2002 working group report: Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin , Working group reports from ITiCSE on Innovation and technology in computer science education*, 35(2):131–152, june 2002.

[159] T.L. Naps, G. Rößling, P. Brusilovsky, J. English, D. Jarc, V. Karavirta, C. Leska, M. McNally, A. Moreno, R.J. Ross, and J. Urquiza-Fuentes. Development of xml-based tools to support user interaction with algorithm visualization. *SIGCSE Bulletin*, 37(4):123–138, 2005.

[160] T.L. Naps, G. Rößling, P. Brusilovsky, J. English, D. Jarc, V. Karavirta, C. Leska, M. McNally, A. Moreno, R.J. Ross, and J. Urquiza-Fuentes. Development of xml-based tools to support user interaction with algorithm visualization. *SIGCSE Bulletin*, 37(4):123–138, 2005.

[161] J. Nielsen. *Usability Engineering*. Academic Press, 1993.

[162] J. Nielsen. *Designing Web Usability*. New Riders Publishing, 1999.

[163] H. Nilsson and J. Sparud. The evaluation dependence tree as a basis for lazy functional debugging. *Automated Software Engineering*, 4(2):121–150, April 1997.

[164] C. North, B. Shneiderman, and C. Plaisant. User controlled overviews of an image library: A case study of the visible human. In *Proceedings of the 1st ACM International Conference on Digital Libraries (DL '96)*, pages 74–82, New York, NY, USA, 1996. ACM Press.

[165] T.S. Norvell and M.P. Bruce-Lockhart. Taking the hood off the computer: Program animation with the teaching machine. In *Proceedings of the Canadian Electrical and Computer Engineering Conference*, pages 831–835, 2000.

[166] Association of Lisp Users. Lisp. http://www.lisp.org, 2007.

[167] M.J. Oudshoorn, H. Widjaja, and S.K. Ellershaw. Aspects and taxonomy of program visualisation. In P. Eades and K. Zhang, editors, *Software Visualisation*, pages 3–26. World Scientific Publishing Co, 1996.

[168] A. Paivio. The empirical case for dual coding. In J.C. Yuille, editor, *Imagery, Memory, and Cognition: Essays in Honor of Allan Paivio*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1983.

[169] C. Pareja-Flores, J. Urquiza-Fuentes, and J.Á. Velázquez-Iturbide. Winhipe: an ide for functional programming based on rewriting and visualization. *ACM SIGPLAN Notices*, 42(3):14–23, March 2007.

[170] C. Pareja-Flores and J.Á. Velázquez-Iturbide. Program execution and visualization on the web. In A. Aggarwal, editor, *Web-Based Learning and Teaching Technologies: Opportunities and Challenges*, pages 236–259. Idea-Group Publishing, Hershey, PA, USA, 2002.

[171] R. Patterson. Hope. http://www.soi.city.ac.uk/∼ross/Hope/, 2007.

[172] K. Perlin and D. Fox. Pad: An alternative approach to the computer interface. In *Proceedings of the 20th Annual ACM Conference on Computer Graphics (SIGGRAPH '93)*, pages 57–64, New York, NY, USA, 1993. ACM Press.

[173] N. Perry. Hope. technical report ic/fpr/lang/2.5. 1/7. Technical report, Dept. of Computing, Imperial College, University of London, UK, October 1989.

[174] W.C. Pierson and S.H. Rodger. Web-based animation of data structures using JAWAA. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, pages 267–271, New York, NY, USA, 1998. ACM Press.

[175] C. Plaisant. The challenge of information visualization evaluation. In *AVI '04: Proceedings of the working Conference on Advanced Visual Interfaces*, pages 109–116, New York, NY, USA, 2004. ACM Press.

[176] C. Plaisant, D. Carr, and B. Shneiderman. Image-browser taxonomy and guidelines for designers. *IEEE Software*, 12(2):21–32, 1995.

[177] S. Pollack and M. Ben-Ari. Selecting a visualization system. In *Proceedings of the Third Program Visualization Workshop*, pages 134–140, Coventry, UK, 2004. University of Warwick.

[178] B. Pope. *Buddha - A Declarative Debbuger for Haskell*. PhD thesis, Department of Computer Science, The University of Melbourne, Australia, june 1998.

[179] K. Powers, S. Ecott, and L.M. Hirshfield. Through the looking glass: teaching CS0 with alice. In *SIGCSE '07: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, pages 213–217, New York, NY, USA, 2007. ACM Press.

[180] B.A. Price, R. Baecker, and I. Small. An introduction to software visualization. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization. Programming as a Multimedia Experience*, pages 3–27. MIT Press, 1998.

[181] J. Rees and W. Clinger. Revised report on the algorithmic language Scheme. *ACM SIGPLAN Notices*, 21(12):37–79, 1986.

[182] C. Reinke. Ghood: Graphical visualisation and animation of haskell object observations. *Electronic Notes in Theoretical Computer Science*, 59:121–149, 2001.

[183] R.A. Rensink. The invariance of visual search to geometric transformation. *Journal of Vision*, 4:178a, 2004.

[184] G.G. Robertson and J.D. Mackinlay. The document lens. In *Proceedings of User Interface Software and Technology '93 (UIST '93)*, pages 101–108, New York, NY, USA, 1993. ACM Press.

[185] G.G. Robertson, J.D. Mackinlay, and S.K. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems '91 (CHI '91)*, pages 189–194, New York, NY, USA, 1991. ACM Press.

[186] J.G. Rokne, B.Wywilla, and X. Wu. Fast line scan-conversion. *ACM Transactions on Graphics*, 9(4):376–388, October 1990.

[187] J.G. Rokne and X. Wu. Double-step incremental generation of lines and circles. *Computer Vision, Graphics and Image Processing*, 37:331–344, 1987.

[188] G-C. Roman and K.C. Cox. Program visualization: The art of mapping programs to pictures. In *International Conference on Software Engineering*, pages 412–420, 1992.

[189] G.-C. Roman and K.C. Cox. A taxonomy of program visualization systems. *IEEE Computer*, 26(12):11–24, 1993.

[190] G.-C. Roman, K.C. Cox, D. Wilcox, and J.Y. Plun. Pavane: a system for declarative visualization of concurrent computations. *Journal of Visual Languages and Computing*, 3:161–193, 1992.

[191] R.J. Ross and M.T. Grinder. Hypertextbooks: Animated, active learning, comprehensive teaching and learning resources for the web. In S. Diehl, editor, *Software Visualization*, volume 2269 of *Lecture Notes in Computer Science*, pages 269–283. Springer-Verlag, Berlin Heidelberg New York, 2001.

[192] G. Rößling and B. Freisleben. Program visualization using AnimalScript. In *First International Program Visualization Workshop*, pages 41–52, Joensuu, Finland, 2001. University of Joensuu Press.

[193] G. Rößling and T.L. Naps. A testbed for pedagogical requirements in algorithm visualizations. *SIGCSE Bulletin*, 34(3):96–100, 2002.

[194] G. Rößling, T.L. Naps, M.S. Hall, V. Karavirta, A. Kerren, C. Leska, A. Moreno, R. Oechsle, S.H. Rodger, J. Urquiza-Fuentes, and J.Á. Velázquez-Iturbide. Merging interactive visualizations with hypertextbooks and course management. In *ITiCSE-WGR '06: Working group reports on ITiCSE on Innovation and technology in computer science education*, pages 166–181, New York, NY, USA, 2006. ACM Press.

[195] G. Rößling and S. Schneider. An integrated and "engaging" package for tree animations. *Electronic Notes in Theoretical Computer Science*, 178:69–78, 2007.

[196] G. Rößling, M Schüer, and B. Freisleben. The animal algorithm animation tool. In *ITiCSE '00: Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEConference on Innovation and Technology in Computer Science Education*, pages 37–40, New York, NY, USA, 2000. ACM Press.

[197] P. Saraiya, C.A. Shaffer, D.S. McCrickard, and C. North. Effective features of algorithm visualizations. In *Proceedings of the Technical Symposium on Computer Sceince Education, SIGCSE'04*, pages 382–386, Norfolk, Virginia, USA, March 2004.

[198] M. Sarkar and M.H. Brown. Graphical fisheye views of graphs. In *CHI '92: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 83–91, New York, NY, USA, 1992. ACM Press.

[199] M. Sarkar and M.H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–84, 1994.

[200] M. Sarkar, S.S. Snibbe, O.J. Tversky, and S.P. Reiss. Stretching the rubber sheet: A metaphor for viewing large layouts on small screens. In *Proceedings of ACM User Interface Software and Technology'93 (UIST '93)*, pages 81–91, New York, NY, USA, 1993. ACM Press.

[201] C.A. Shaffer, L.S. Heath, and J. Yang. Using the Swan data structure visualization system for computer science education. In *SIGCSE '96: Proceedings of the twenty-seventh SIGCSE Technical Symposium on Computer Science Education*, pages 140–144, New York, NY, USA, 1996. ACM Press.

[202] A.T.M. Shafiqul-Khalida and M. Kaykobad. An efficient line algorithm. In *IEEE 39th Midwest Symphosium on Circuits and Systems*, volume 3, pages 1280–1282, Dayton, OH, 1996.

[203] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *VL '96: Proceedings of the 1996 IEEE Symposium on Visual Languages*, page 336, Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[204] B. Shneiderman. *Designing the User Interface*. Addison-Wesley, Reading, MA, USA, 1998.

[205] B. Shneiderman and C. Plaisant. *Diseño de Interfaces de Usuario. Estrategias para una interacción Persona-computadora Efectiva*. Pearson/Addison-Wesley, Madrid, Spain, 2006.

[206] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, 1994.

[207] T.R.H. Sizer. *The Digital Differential Analyser*. Chapman and Hall Ltd., London, UK, 1968.

[208] J. Sparud and C. Runciman. Tracing lazy functionals computations using redex trails. In H. Glasser, P. Hartel, and H. Kuchen, editors, *Proceedings of the 9th International Symposium on Programming Languages, Implementations, Logics and Programs (PLILP'97)*, volume 1292 of *Lecture Notes in Computer Science*, pages 291–308. Springer-Verlag, Berlin, Germnay, 1997.

[209] R. Spence. *Information Visualization*. Addison Wesley / ACM Press, 2001.

[210] R. Spence and M.D. Apperley. Data base navigation: an office environment for the professional. In S.K. Card, J.D. Mackinlay, and B. Shneiderman, editors, *Information Visualization: Using Vision to Think*, pages 331–340. Morgan Kaufmann Publishers, 1999.

[211] J. Stasko and J. Wehrli. Three-dimensional computation visualization. In *Proceedings of the 1993 IEEE Symphosium on Visual Languages*, pages 100–107, Los Alamitos, CA, USA, August 1993. IEEE Computer Society Press.

[212] J.T. Stasko. Animating algorithms with XTANGO. *SIGACT News*, 23(2):67–71, 1992.

[213] J.T. Stasko. Using student-built algorithm animations as learning aids. In *SIGCSE '97: Proceedings of the twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, pages 25–29, New York, NY, USA, 1997. ACM Press.

[214] J.T. Stasko, A. Badre, and C. Lewis. Do algorithm animations assist learning?: an empirical study and analysis. In *CHI '93: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 61–66, New York, NY, USA, 1993. ACM Press.

[215] J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price. *Software Visualization. Programming as a Multimedia Experience*. MIT Press, 1998.

[216] J.T. Stasko and E. Kraemer. A methodology for building application-specific visualisations of parallel programs. *Journal of Parallel and Distributed Computing*, 18:258–264, 1993.

[217] J.T. Stasko and C. Patterson. Understanding and characterizing software visualization systems. In *Proceedings of the 1992 IEEE Workshop on Visual Languages*, pages 3–10, 1992.

[218] L. Stern, H. Søndergaard, and L. Naish. A strategy for managing content complexity in algorithm animation. In *Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, pages 127–130, New York, NY, USA, 1999.

[219] G. Strang and G. Fix. A fourier analysis of the finite element variational method. In *Constructive Aspects of Functional Analysis*, pages 795–840. Edizioni Cremonese, 1973.

[220] E. Sutinen, J. Tarhio, and T. Tërasvirta. Easy algorithm animation on the web. *Multimedia Tools and Applications*, 19:179–194, 2003.

[221] J.P. Taylor. *Presenting the lazy evaluation of functions*. PhD thesis, Department of Computer Science, Queen Mary University of London and Westfield College, London, UK, 1995.

[222] M. Tory and T. Moller. Rethinking visualization: A high-level taxonomy. In *Proceedings of IEEE Symposium on Information Visualization. INFOVIS 2004*, pages 151–158, Los Alamitos, CA, USA, 2004. IEEE Computer Society Press.

[223] D.S. Touretzky and P. Lee. Visualizing evaluation in applicative languages. *Communications of the ACM*, 35(10):49–59, october 1992.

[224] A. Treisman. Preattentive processing in vision. *Computer Vision, Graphics, and Image Processing*, 31(2):156–177, 1985.

[225] E.R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1983.

[226] D.A. Turner. Miranda: a non-strict functional language with polymorphic types. In *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*, pages 1–16, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[227] J. Urquiza-Fuentes. Evaluación de niveles de implicación de los estudiantes con la visualización de programas funcionales. Seminario de Investigación en Tecnologías de la Información Aplicadas a la Educación, SITIAE 2007, 2007.

[228] J. Urquiza-Fuentes, C.A. Lázaro-Carrascosa, and J.Á. Velázquez-Iturbide. Improving a zoom+pan interface with overview+detail or focus+context features: a comparative evaluation. In M.A. Redondo, C. Bravo, and M. Ortega, editors, *Engineering the User Interface*, page In press. Springer-Verlag, Berlin Heidelberg New York, 2008.

[229] J. Urquiza-Fuentes and J.Á. Velázquez-Iturbide. Effortless construction and management of program animations on the web. In *Advances in Web-Based Learning - ICWL 2005: Fourth International Conference*, volume 3583 of *Lecture Notes in Computer Science*, pages 163–173, Berlin, Germany, 2005. Springer-Verlag.

[230] J. Urquiza-Fuentes and J.Á. Velázquez-Iturbide. R-Zoom: A visualization technique for algorithm animation construction. In *Proceedings of the IADIS International Conference Applied Computing 2005*, pages 145–152, Lisboa, Portugal, 2005. IADIS Press.

[231] J. Urquiza-Fuentes and J.Á. Velázquez-Iturbide. An evaluation of the effortless approach to build algorithm animations with winhipe. *Electronic Notes in Theoretical Computer Science*, 178:3–13, July 2007.

[232] J. Urquiza-Fuentes, J.Á. Velázquez-Iturbide, and C.A. Lázaro-Carrascosa. Design and evaluation of R-Zoom, a new focus+context visualization technique. In *INTERACCIÓN 2006 Diseño de la Interacción Persona-Ordenador: Tendencias y Desafíos*, pages 91–100, Ciudad Real, España, 2006. Universidad de Castilla la Mancha.

[233] J.Á. Velázquez-Iturbide. Improving functional programming environments for education. In M.D. Brouwer-Janse and T.L. Harrington, editors, *Man-Machine Communication for Educational Systems Design*, pages 325–332. Springer-Verlag, Berlin Heidelberg, Germany, 1994.

[234] J.Á. Velázquez-Iturbide. Automatic simplification of the visualization of functional expressions by means of fisheye views. In *Proceedings of the Joint Conference on Declarative Programming (APPIA-GULP-PRODE'98)*, pages 101–112, La Coruña, España, 1998.

[235] J.Á. Velázquez-Iturbide. Principled design of logical fisheye views of functional expressions. *ACM SIGPLAN Notices*, 41(8):34–43, August 2006.

[236] J.Á. Velázquez-Iturbide, C. Pareja-Flores, and J. Urquiza-Fuentes. An approach to effortless construction of program animations. *Computers & Education*, 50(1):179–192, january 2008.

[237] J.Á. Velázquez-Iturbide, D. Redondo-Martín, C. Pareja-Flores, and J. Urquiza-Fuentes. An instructor's guide to design web-based algorithm animations. In *Advances in Web-Based Learning - ICWL 2007: Sixth International Conference*, volume In Press of *Lecture Notes in Computer Science*, pages 163–173, Berlin, Germany, 2007. Springer-Verlag.

[238] J.Á. Velázquez-Iturbide, Á. Sánchez-Calle, A. Duarte-Muñoz, and C.A. Lázaro-Carrascosa. *Ejercicios Resueltos de Bases de Lenguajes de Programación*. Editorial universitaria Ramón Areces, Madrid, España, 2005.

[239] J.Á. Velázquez-Iturbide and A. Vázquez-Presa. Customization of visualizations in a functional programming environment. In *Proceedings of the 29th ASEE/IEEE Frontiers in Education Conference*, pages 22–28, Champaign, IL, USA, 1999. Stipes Publishing.

[240] A.T. Virtanen, E. Lahtinen, and H.-M. Järvinen. VIP, a visual interpreter for learning introductory programming with C++. In *Proceedings of The Fifth Koli Calling Conference on Computer Science Education*, pages 125–130, 2005.

[241] C. Ware. *Information Visualization. Perception for Design*. Morgan Kaufmann Publishers, 2000.

[242] R. Watson and E. Salzman. A trace browser for a lazy functional language. In *Proceedings of the Twentieth Australasian Computer Science Conference*, pages 356–363, 1997.

[243] S. Wehrend and C. Lewis. A problem-oriented classification of visualization techniques. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'90)*, pages 139–143, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[244] A. Woodruff, J. Landay, and M. Stonebreaker. Goal-directed zoom. In *Summary of the ACM Conference on Human Factors in Computing Systems (CHI '98)*, pages 305–306, New York, NY, USA, 1998. ACM Press.

# Glosario

**Animación** Ver *Visulización dinámica*. En el ámbito de esta tesis se referiere específicamente a la visualización dinámica del comportamiento de un programa o algoritmo, 53

**Animación post-mortem** Aquellas animaciones cuya construcción y reproducción se producen una vez que el programa correspondiente ha finalizado su ejecución, 46

**Constructivismo** Teoría según la cual, el aprendizaje es un proceso social que da sentido a la experiencia en términos del conocimiento existente. Todo conocimiento es construido de forma activa por el propio estudiante. El papel del profesor pasa a la creación de situaciones donde los estudiantes puedan aprender los conceptos lo mejor posible construyendo su propio conocimiento [71], 107

**Contexto** En las interfaces *Focus+Context*, los elementos en los que el usuario no está especialmente interesado, 74

**Eficacia** La corrección y/o completitud con que los usuarios alcanzan los objetivos [98], 89

**Eficiencia** Los recursos utilizados, en relación con la eficacia con que los usuarios alcanzan los objetivos [98], 90

**Elementos (del espacio de trabajo)** Información que el usuario observa y manipula con la técnica de visualización, 72

**Espacio de trabajo** Zona que contiene la información con la que el usuario ha de trabajar, 72

**Evaluación (paradigma de programación funcional)** Proceso de ejecución de un programa funcional, 14

**Expresión (paradigma de programación funcional)** Secuencia de operadores y operandos. La aplicación de los operadores a los operandos producen a su vez nuevos operandos, siendo este el proceso básico de ejecución de un programa funcional, 14

**Facilidad de uso** Característica que indica cómo de complejo es para un usuario utilizar la aplicación, 92

**Factor intermedio** En investigación educativa, se llaman factores intermedios a las variables que pueden afectar al tratamiento o los individuos de un experimento, pero que no son controlables por el investigador, [48], 119

**Foco** En las interfaces *Focus+Context*, el elemento en que el usuario está interesado, su representación ocupará más espacio para mostrar más detalle, 74

**Focus+Context** Familia de técnicas de visualización de la información que integra en una sola vista del espacio de trabajo, información con diferentes grados de detalle. La información asociada al *foco* tendrá más detalle que aquella asociada al contexto, 74

**Hojas de estilo CSS** Cascading Style Sheets (CSS) [49], es un mecanismo que permite añadir estilos de presentación a las páginas Web, independizando el contenido de su presentación, 102

**HTML** HyperText Markup Language [52], leguaje que sirve para construir documentos publicables en la Web, 98

**Interpolación (tratamiento digital de la imagen)** Operación por la que se construyen nuevos píxeles en función de los originales, 58

**Interpolación de grado 0** Interpolación que utiliza la *regla de vecindad* para la generación de nuevos píxeles en una imagen. La regla de vecindad establece que el píxel nuevo es aquel más cercano a su posición en la imagen original, 59

**Interpolación de grado superior** Interpolación que utiliza polinomios de grado superior a 1, para aproximar los píxeles originales, 58

**JavaScript** Lenguaje interpretado, con sintaxis similar a java, usado para proporcionar capacidades dinámicas a las páginas Web escritas en HTML, 98

**Marcos embebidos** InLine frames [51], representan una evolución de los marcos normales, permitiendo insertar dentro de elementos de páginas Web otros documentos sin necesidad de crear esqueletos previos, 102

**Miniatura** En el ámbito de esta tesis se refiere específicamente a la versión reducida, en tamaño, de una visualización, 53

**Overview+Detail** Familia de técnicas de visualización de la información basada dos vistas sincronizadas del espacio de trabajo, una vista muestra una visión global (overview) del mismo, mientras que la otra muestra una visión detallada (detail) de una parte del espacio de trabajo, 73

**Preferencia del usuario** Ver *Satisfacción del usuario*, 92

**Pretty-printing** La manipulación del formato (fuente, tamaño, color, ...) de un texto para resaltarlo del resto de contenidos presentados, 12

**Redex** Parte de la expresión funcional que será evaluada en el siguiente paso de reescritura, 14

**Relación de aspecto** La proporción existente entre la altura y la anchura de una imagen, 56

**Satisfacción del usuario** Opinión subjetiva del usuario respecto de la aplicación, podría resumirse en la pregunta *¿Le gusta la aplicación?*[161]. También se define como la comodidad con que se siente el usuario al trabajar con la interfaz, así como la aceptación o rechazo a usarlo [98], 91

**Taxonomía de Bloom** Taxonomía que clasifica los objetivos educativos en 6 niveles organizados de forma jerárquica: conocimiento, comprensión, aplicación, análisis, síntesis y evaluación [28], 26

**Usabilidad** Característica que describe en qué medida, un producto se puede usar por determinados usuarios para conseguir ciertos objetivos con eficacia, eficiencia y satisfacción en un contexto específico de uso [98], 85

**Utilidad** Característica que indica si determinada aplicación proporciona a los usuarios las herramientas necesarias para realizar las tareas indicadas, 92

**Visualización** Representar datos por medio del ordenador, de forma visual e interactiva, para ampliar el conocimiento, [42]. En el ámbito de esta tesis se refiere específicamente a las representaciones visuales de los estados por los que pasa la ejecución de un programa o algoritmo, 9

**Visualización de algoritmos** Visualizaciones de alto nivel que describen el software. [180], 12

**Visualización de la información** Representar datos abstractos por medio del ordenador, de forma visual e interactiva, para ampliar el conocimiento, [42], 9

**Visualización de programas** La presentación gráfica, monitorización y exploración de programas expresados de forma textual. [188], 12

**Visualización del software** El uso de las artes de tipografía, diseño gráfico, animación y cinematografía con las tecnologías modernas de interacción persona-ordenador e informática gráfica para facilitar tanto la comprensión humana como el uso eficaz del software de ordenador [180], 12

**Visualización dinámica** Representación visual (ver el término Visualización) que evoluciona y cambia a lo largo del tiempo, también llamada animación, 12

**Visualización estática** Representación visual (ver el término Visualización) fija que no cambia ni evoluciona, independientemente del contenido representado, 12

**XHTML** Lenguaje de marcado de propósito general que sirve para crear documentos publicables en la Web, también es la aplicación de los principios de XML a HTML, 99

**XML** eXtensible Markup Language [50], es un formato de texto que permite construir documentos semiestructurados, se ha desarrollado a partir de SGML [97], 102

**XSLT** XSL Transformations [54], lenguaje que sirve para transformar documentos xml en otros documentos xml, en nuestro caso transformamos xml en xhtml [53], 102

**Zoom semántico** Técnica de visualización que permite mostrar mayor o menor cantidad de contenidos de un objeto según sea el grado de detalle con que se muestra. Por ejemplo, en un documento mostrado con poco grado de detalle, sólo se vería el título del documento y de cada

una de las secciones. Aumentando el grado de detalle, aumentarían la información mostrada a los contenidos de las secciones, 72

**Zoom+Pan** Familia de técnicas de visualización de la información basada en el desplazamiento por el espacio de trabajo permitiendo mostrar con mayor o menor detalle la información, 72