UNA INNOVACIÓN LITERARIA EN LA ENSEÑANZA DE LA PROGRAMACIÓN

Manuel Perera Domínguez
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

RESUMEN

Es un hecho admitido que la documentación de los programas es usualmente muy pobre y poco comprensible para las personas que no han participado en su desarrollo lo que dificulta enormemente las tareas de mantenimiento del software y su reutilización posterior. La escasa legibilidad de los programas limita en gran medida el uso docente de los listados de código fuente como medio de aprendizaje de la programación a través de ejemplos. La "programación literaria" (*literate programming*) permite redactar programas más legibles, mantenibles, reusables y portables, adecuados para uso docente y correctamente documentados.

ABSTRACT

It's a very well known fact that computer program documentation is usually very poor, hardly comprehensible, and little useful. This greatly difficults the maintenance of software and its reusability. The teaching of programming by the "learning from examples" methodology is seriously limited by problems in program understanding and code reading. Literate programming improves program comprehension and generates elegantly documented and structured programs which are very well suited for teaching.

LEGIBILIDAD DE LOS PROGRAMAS DE ORDENADOR

Desde los años 50 hasta la actualidad la edición y publicación del código fuente de los programas de ordenador ha permanecido esencialmente invariable: un listado completo del código tal y como será compilado al que se han añadido comentarios explicativos. Los programas suelen ser difíciles de entender como lo prueba el que apenas aparezcan grandes listados en los textos universitarios de informática. Dos ejemplos extensos y complejos de lo máximo que puede conseguirse con las técnicas tradicionales de documentación de código se encuentran en conocidos textos académicos sobre sistemas operativos [12] y compiladores [5]. Muchas veces los programas son más difíciles de leer que de escribir.

En la práctica profesional de la programación se concede gran importancia a la lectura de programas: *"...creo que uno de los mejores tests de capacidad de programación consiste en entregar al programador unas treinta páginas de código y ver con qué velocidad es capaz de leerlas y comprenderlas ... la mejor forma de prepararse es escribir programas y estudiar grandes programas que hayan escrito otros ... Uno tiene que estar siempre dispuesto a leer código escrito por otros, luego escribir el suyo propio y luego dárselo a revisar a otras personas"* (Bill Gates en [11] pág. 94). Esta importancia también ha sido reconocida por la comunidad científica que plantea la conveniencia del aprendizaje de la programación a través de ejemplos: un buen programa debe enseñar al lector. Citemos a N. Wirth en su clásica obra "Algoritmos + estructuras de datos = programas": *"Estos ejemplos están también pensados como ejercicios de aprendizaje de la lectura de programas que demasiado frecuentemente se olvida en favor de la escritura... lo que permanece del*

arsenal de métodos y enseñanza es la selección y presentación cuidadosa de ejemplos clave" ([14] pág. XV). Para observar cómo la falta de legibilidad dificulta el uso docente de programas de ejemplo véase el deficiente libro de soluciones [13] de los problemas del conocido texto de Kernighan y Ritchie sobre el lenguaje de programación C [6].

Por supuesto la legibilidad es muy importante para un estudiante de programación pues un profesor humano ha de leer y comprender sus programas para puntuarlos [2].

MANTENIBILIDAD Y REUSABILIDAD DEL SOFTWARE

El que un programa de ordenador sea legible es condición necesaria para que éste sea mantenible y reutilizable, sobre todo si grupos diferentes de personas redactan y mantienen el software. Los estudios de ingeniería del software indican que el tiempo necesario para entender código representa muchas veces más del 50% del coste de mantenimiento. La escasa legibilidad del código y la falta de una documentación actualizada es lo habitual por lo que muchas veces el mantenimiento degenera en una especie de "programación experimental" en la que no se entiende bien el programa y no se sabe cómo afectarán los cambios. Puede generarse código ya existente, eliminar o introducir dependencias desconocidas, etc. en un frenético proceso de agregación de código nuevo, ejecución del programa y observación de los nuevos errores producidos. Un programa mal documentado y poco legible será casi con toda seguridad desechado si su programador abandona la empresa u organismo que lo desarrolló.

PROGRAMACIÓN LITERARIA

La idea central de la programación literaria es que los programas deberían ser escritos para que sean entendibles por un humano y no para que sean directamente legibles por un compilador. Para ello se aúnan las labores de programación y documentación, tradicionalmente separadas, utilizando conjuntamente un lenguaje de programación y un lenguaje de descripción textual. El programa se considera como un documento estructurado dirigido hacia un lector humano por lo que se fomenta una exposición clara de las ideas del programador y un cierto estilo "literario" de escritura, algo totalmente inexistente en los típicos y crípticos listados de programas solo entendibles por un experto. Un programa literario puede incluir tablas, figuras, referencias bibliográficas, ecuaciones, índices automáticos, etc. para lograr una documentación de muy alta calidad que permita percibir la estructura de una compleja pieza de software. Para facilitar la comprensión del documento (programa) éste se estructura en pequeñas unidades conceptuales interrelacionadas entre sí de manera que la apariencia final de un programa literario es la de un documento en el que se ha intercalado código fuente, justo al contrario de lo habitual. La programación literaria es en principio adaptable a cualquier lenguaje de programación y no restringe la libertad del programador si bien de manera natural fomenta un enfoque estructurado, descendente (*top-down*) y de refinamiento sucesivo.

ENSEÑANZA LITERARIA DE LA PROGRAMACIÓN

La enseñanza literaria de la programación persigue los siguientes objetivos:

- Los estudiantes deben comprender la importancia de la documentación para redactar programas legibles y mantenibles. La documentación ha de ser parte intrínseca del programa y no un añadido improvisado en el último momento como es actualmente.

- Cambiar la actitud del estudiante: su tarea principal no ha de ser instruir a un computador sobre qué hacer sino poder comunicar a otro humano qué quiere que haga el programa. Ha de ser tan importante "comunicar" con las personas que lean el programa como con el compilador del lenguaje de programación.
- Desarrollar el trabajo cooperativo en pequeños grupos. Los estudiantes deben leer y revisar el código redactado por sus compañeros, señalando posibles defectos y mejoras y aprendiendo de sus aciertos en un ciclo de realimentación positiva.
- Considerar un programa como un diseño dirigido a la resolución de un problema y que se plantea en términos de objetivos para los que ha de crearse código para cumplirlos.
- Mostrar el método por el cual se derivó la solución del problema. La mayor parte de los libros de texto sobre programación dan soluciones "totales" sin mostrar el proceso de diseño. En este sentido la programación literaria refuerza el papel del diseño y permite ver el programa como una consecuencia de unas decisiones de diseño bien documentadas.
- Obligar a especificar qué hace el programa en su totalidad y cada parte del mismo. Esto permite comprobar si los estudiantes han entendido bien el problema que han de resolver. Es una actitud muy común ante un problema de programación el empezar a codificar de inmediato y ejecutar repetidas veces observando los errores producidos. Muchas veces los errores son debidos a una inadecuada comprensión del problema planteado y a que el estudiante no sabe (no es capaz de explicar) qué hace exactamente el código que desarrolla y por qué. Se ha de procurar reducir en lo posible el uso irreflexivo de un depurador (*debugger*).
- Ayudar a desarrollar programas claros, limpios, bien estructurados y coherentes.

Los lenguajes de programación se dirigen fundamentalmente a obtener código legible por los compiladores, lo cual obliga a una serie de convenciones sintácticas que dificultan la comprensión de los programas. El listado de un programa se presenta y se lee de manera lineal en un orden que no coincide con el de escritura: un programador no escribe desde la primera hasta la última línea de código "de una vez" sino que realiza varios ciclos de correcciones, vueltas atrás, saltos hacia adelante, etc. Desde el punto de vista de la comprensión de un programa es mucho más conveniente presentarlo siguiendo su orden lógico de desarrollo, prescindiendo en la medida de lo posible de las limitaciones sintácticas del lenguaje de programación empleado. Hay que evitar que los estudiantes organicen su conocimiento recién adquirido de programación en términos de la sintaxis del lenguaje concreto que utilicen. Los programadores expertos organizan su contenido en unidades conceptuales que transcienden la sintaxis concreta del lenguaje de programación y que pueden encontrar una expresión natural en un programa literario.

SISTEMAS DE PROGRAMACIÓN LITERARIA

En la práctica un sistema de programación literaria debe permitir lo siguiente:

- Generar conjuntamente documentación y código en un lenguaje de programación de alto nivel que residirán en un mismo fichero. La unión de documentación y código es un programa literario. Resulta mucho más fácil el ajustar la documentación a los cambios realizados en el código evitando su desfase e inutilidad.
- Especificar subdivisiones lógicas en el programa (módulos, secciones, etc.) que pueden interrelacionarse. Cada sección lógica consta de una parte de documentación y otra de código fuente.

- Escribir el programa en un orden lógico sin restricciones sintácticas. Las partes del programa deben presentarse en un orden que ayude a entender cómo funciona éste, que no es necesariamente el orden requerido por el compilador.
- Extraer automáticamente a partir del programa literario el código fuente del programa en el orden requerido por el compilador del lenguaje empleado.
- Extraer automáticamente la documentación a partir del programa literario. La documentación debe explicitar claramente lo que hace el programa y los medios de solución empleados e incluye el código fuente. Pueden usarse fórmulas matemáticas, tablas, gráficos y enlaces hipertextuales para ayudar a su comprensión. Esta documentación debe ser tan extensa como se precise y puede incluir detalles que habitualmente no están presentes en el código de un programa: consideraciones de diseño, discusión de soluciones alternativas, pruebas de ejecución, etc. que pueden resultar de interés.
- Generar un documento final (programa literario) de la máxima calidad visual para lo cual se utilizarán técnicas de *pretty printing* pues está demostrado que el resalte tipográfico ayuda a la comprensión de los programas [1].

El primer sistema de programación literaria fue "Web", desarrollado por Donald Knuth (considerado por muchos como el mejor programador del mundo) cuando tuvo que reescribir TeX. Knuth ha empleado extensivamente la programación literaria [9] con la que ha desarrollado programas muy portables y extensamente utilizados [7,8,10]. En la actualidad existen numerosos sistemas de programación literaria (CWeb, Noweb, etc.) aptos para una gran diversidad de lenguajes (Pascal, C, C++, Fortran, Modula 2, Ada, Lisp, Prolog, Maple, APL, etc) y que se diferencian en el grado de dependencia que presentan respecto al lenguaje de programación concreto y el formateador de texto empleado (usualmente LaTeX). Todos ellos tienen en común unas herramientas (procesadores) genéricamente denominadas "Tangle" y "Weave". La primera de ellas reordena el código fuente presente en el programa literario de acuerdo con el orden lógico seguido y produce el fichero que será finalmente compilado. Internamente funciona mediante expansiones de macros. La herramienta Weave genera la documentación del programa literario (su "listado"). En [3] y [4] pueden examinarse programas complejos y muy portables desarrollados en la comunidad académica utilizando programación literaria.

EJEMPLO

A modo de ejemplo se incluye un programa literario para ilustrar el algoritmo de ordenación de inserción directa. El programa se ha diseñado y descrito jerárquicamente siguiendo la metodología de refinamiento sucesivo propuesta por N. Wirth [14] que permite abstraer gradualmente los detalles y presentar un pseudocódigo estructurado y muy legible. El resultado es un código que puede ser leído secuencialmente como un libro.

Ejemplo de programación literaria: ordenación por inserción directa

A Introducción

Este programa ordena ascendentemente los elementos de un vector de números enteros utilizando el algoritmo de inserción directa. Se trata de un método directo de ordenación ya que las permutaciones de elementos se realizan "in situ". Se ha empleado el lenguaje de programación C. Recuérdese que en C a la primera posición de un vector le corresponde el

índice "0" y no el "1". Este método de ordenación no realiza intercambios entre posiciones del vector a ordenar al contrario que los métodos de burbuja y selección directa.

B Ordenación por inserción directa

Entrada:
- Vector t de N números enteros.
- Posiciones inicial a y final b del subvector a ordenar ($0 \leq a \leq b < N$).

Resultado:
- Los elementos del subvector $t_a,...,t_b$ han sido permutados de manera que cada elemento es menor o igual que el siguiente: $t_i \leq t_{i+1}$ con $a \leq i < b$.

La idea básica consiste en considerar el vector t como ya ordenado en sus i primeras posiciones ($t_0,..., t_{i-1}$ es pues un subvector ordenado) e insertar el elemento de la posición i en el lugar adecuado del subvector ordenado de manera que resulten ordenados los primeros i+1 elementos, incrementándose en una unidad el tamaño del subvector ordenado. Se trata pues de un proceso iterativo en el que incrementando el valor de i se acaba por ordenar todo el subvector.

C Programa de ejemplo

1a ⟨*programa de ejemplo del método de ordenación por inserción directa* 1a⟩≡
```
⟨includes del programa 1b⟩
⟨definiciones del programa 2a⟩
void main()
{   ⟨declarar variables 2b⟩
    ⟨inicializar variables 2c⟩
    ⟨ordenar el subvector t_a,...,t_b por inserción directa 3a⟩
    ⟨comprobar la ordenación del subvector 4a⟩}
```

1b ⟨*includes del programa* 1b⟩≡ (1a) 2g ▷
```
#include <stdio.h>
```

Supóngase que el vector tendrá como máximo N = 100 elementos.

2a ⟨*definiciones del programa* 2a⟩≡ (1a) 2f ▷
```
#define N 100
```

Definición del vector t y de los límites a y b del subvector a ordenar.

2b ⟨*declarar variables* 2b⟩≡ (1a) 2e ▷
```
int t[N], a, b;
```

Se inicializa el vector con números enteros aleatorios en el intervalo $Z_1,...,Z_2$. Previamente hay que inicializar el generador de números aleatorios. Se ordenará todo el vector por lo que se toma $a = 0$ y $b = N$-1.

2c ⟨*inicializar variables* 2c⟩≡ (1a)
⟨*inicializar generador de números aleatorios* 2d⟩
for(i←0; i<N; i++)
$t_i \leftarrow Z_1 + (\textbf{rand}() \% (Z_2 - Z_1 + 1))$;
a←0; b←N-1;

Como valor de semilla inicial para el generador de números aleatorios se utiliza el número que el sistema operativo asigna al programa o proceso y que varía impredeciblemente de una ejecución a otra del programa.

2d ⟨*inicializar generador de números aleatorios* 2d⟩≡ (2c)
srand(getpid());

Para inicializar, ordenar y comprobar la ordenación del subvector es necesario declarar una variable índice entera i.

2e ⟨*declarar variables* 2b⟩+≡ (1a)◁ 2b 3f ▷
int i;

Supóngase $Z_1 = 1$ y $Z_2 = 1.000$ (límites de los números aleatorios).

2f ⟨*definiciones del programa* 2a⟩+≡ (1a)◁ 2a
#define Z_1 1
#define Z_2 100

Las funciones de números aleatorios se definen en el fichero de cabecera <stdlib.h> de la biblioteca estándar de C.

2g ⟨*includes del programa* 1b⟩+≡ (1a)◁ 1b
#include <stdlib.h>

El proceso iterativo de ordenación empieza por el segundo elemento del subvector (pues evidentemente el primero considerado aisladamente ya está ordenado) y finaliza en el último elemento del subvector.

3a ⟨*ordenar el subvector* $t_a,...,t_b$ *por inserción directa* 3a⟩≡ (1a)
for(i←a+1; i≤b; i++)
{ ⟨*insertar* t_i *en el subvector ordenado* $t_a,...,t_{i-1}$ 3b⟩ }

El subvector de destino $t_a,...,t_{i-1}$ ya está ordenado ascendentemente por lo que para insertar el elemento $x = t_i$ se puede utilizar el esquema de inserción binaria o dicotómica que es más eficiente que la inserción lineal.

3b ⟨*insertar* t_i *en el subvector ordenado* $t_a,...,t_{i-1}$ 3b⟩≡ (3a)
x←t_i; izq←a; der←i-1;
while(izq≤der)

{ ⟨*reducir intervalo* 3c⟩ }
⟨*hacer espacio para insertar* t_i 3d⟩
⟨*insertar* t_i 3e⟩

El proceso arranca del elemento central del subvector de destino y continúa por bisección hasta encontrar el punto de inserción. Si se comparase con ≤ la ordenación sería inestable.

3c ⟨*reducir intervalo* 3c⟩≡ (3b)
```
mitad←(izq+der)/2;
if(x<t_mitad)
    der←mitad-1;
else
    izq←mitad+1;
```

El elemento $x = t_i$ se acabará insertando en la posición indicada por *izq* para lo cual hay que desplazar una posición a la derecha el contenido de las posiciones $t_{izq},\ldots,t_{i-1}$. Nótese que en el proceso se pierde el contenido original de t_i razón por la que fue necesario anteriormente almacenarlo en la variable x con lo que además se gana en eficiencia ya que en cada paso de la inserción binaria no hay que acceder al vector a través del índice i sino que basta con consultar el valor de la variable x.

3d ⟨*hacer espacio para insertar* t_i 3d⟩≡ (3b)
```
for(j←i; j>izq; j--) t_j ← t_{j-1};
```

3e ⟨*insertar* t_i 3e⟩≡ (3b)
```
t_izq ← x;
```

Hay que definir las variables enteras *izq*, *der*, *mitad*, x y j.

3f ⟨*declarar variables* 2b⟩+≡ (1a)◁ 2e
```
int izq, der, mitad, x, j;
```

Por último se realiza un recorrido del subvector para comprobar que está realmente ordenado. Si se encuentra un elemento que es menor que el colocado en la posición anterior puede concluirse que el vector no está ordenado ascendentemente.

4a ⟨*comprobar ordenación del subvector* 4d⟩≡ (1a)
```
for(i←a+1; i≤b; i++) if(t_i < t_{i-1}) break;
if(i=b+1) printf("Vector ordenado ascendentemente.\n");
else printf("Vector no ordenado ascendentemente.\n");
```

D Índice

⟨*comprobar ordenación del subvector* 4a⟩
⟨*declarar variables* 2b⟩
⟨*definiciones del programa* 2a⟩
⟨*hacer espacio para insertar* t_i 3d⟩

⟨*includes del programa* 1b⟩
⟨*inicializar generador de números aleatorios* 2d⟩
⟨*inicializar variables* 2c⟩
⟨*insertar* t_i 3e⟩
⟨*insertar* t_i *en el subvector ordenado* $t_a,\ldots,t_{i-1}$ 3b⟩
⟨*ordenar el subvector* $t_a,\ldots,t_b$ *por inserción directa* 3a⟩
⟨*programa de ejemplo del método de ordenación por inserción directa* 1a⟩
⟨*reducir intervalo* 3c⟩

a: 2b, 2c, 3a, 3b, 4a
b: 2c, 2c, 3a, 4a
der: 3b, 3c, 3f
i: 2c, 2e, 3a, 3b, 3d, 4a
izq: 3b, 3c, 3d, 3e, 3f
j: 3d, 3f
mitad: 3c, 3f
N: 2a, 2b, 2c
t: 2b, 2c, 3b, 3c, 3d, 3e, 4a
x: 3b, 3c, 3e, 3f
Z_1: 2c, 2f
Z_2: 2c, 2f

La herramienta Tangle procesa el programa literario y extrae el código fuente en lenguaje C en el orden apropiado procediendo a partir de la sección inicial 1a. El código resultante es la concatenación del código incluido en las secciones 1b, 2g, 2a, 2f, 2b, 2e, 3f, 2d, 2c, 3a, 3b, 3c, 3d, 3e y 4a.

```
#include <stdio.h>
#include <stdlib.h>
#define N 100
#define Z1 1
#define Z2 1000
void main()
{
 int t[N], a, b;
 int i;
 int izq, der, mitad, x, j;
 srand(getpid());
 for(i=0; i<N; i++)
  t[i] = Z1 + (rand() % (Z2-Z1+1));
 a=0; b=N-1;
 for(i=a+1; i<=b; i++) {
  x = t[i]; izq = a; der = i-1;
  while( izq <= der) {
```

```
      mitad = (izq + der) / 2;
      if(x < t[mitad])
        der = mitad - 1;
      else
        izq = mitad + 1;
      }
    for(j=i; j>izq; j--) t[j] = t[j-1];
    t[izq] = x;
    }
  for(i=a+1; i<=b; i++)
    if(t[i] < t[i-1])
      break;
  if(i == b+1)
    printf("Vector ordenado ascendentemente.\n");
  else
    printf("Vector no ordenado ascendentemente.\n");
}
```

DISCUSIÓN DEL EJEMPLO

La documentación generada es sin duda muy atractiva. Se observa claramente la estructura jerárquica de secciones de código del programa y la estructuración lógica que ha producido la aplicación de la metodología de refinamiento sucesivo. En el ejemplo las secciones de código se denotan en función de la página del documento en que aparecen de manera que la sección 2c es la que aparece en la segunda página del documento en tercer lugar. Las secciones se referencian unas a otras automáticamente lo que facilita la percepción clara de la estructura del programa. Por ejemplo la sección 2f es una ampliación de la sección ⟨*definiciones del programa* 2a⟩ y es invocada desde la sección inicial 1a.

El código fuente insertado en el programa literario es código C puro y como tal es extraído por la herramienta Tangle pero en el documento producido por Weave se perciben importantes diferencias:

- Se resaltan tipográficamente los nombres de variables, funciones y palabras reservadas del lenguaje de programación.
- No se utilizan los operadores de asignación = ni de comprobación de igualdad == del lenguaje C sino los símbolos universalmente empleados en los pseudocódigos: ← y =.
- La indexación de los elementos de la tabla utiliza una notación matemática con subíndice: t_i en vez de t[i]. De manera similar los operadores relacionales se escriben como es habitual en matemáticas (≤ en vez de <=, etc.)

Todo lo anterior conduce a un código más universal y cercano al pseudocódigo e independiente de determinados aspectos sintácticos del lenguaje de programación empleado. Al final del programa se incluye un índice con todas las secciones del programa y una lista de identificadores que informa de la sección en que se definió cada identificador y de las secciones que lo utilizan. Fácilmente puede "seguirse la pista" de una variable a lo largo del programa, controlar cuándo se modifica su valor, etc.

Obsérvese que el orden del código expuesto en el programa literario no es el mismo que el del código C extraído por la herramienta Tangle. En el programa literario las variables se han definido cuando aparecen por primera vez y no antes (véase por ejemplo la variable *i*) y algo similar sucede con los ficheros de cabecera (no se incluye <stdlib.h> hasta que aparecen los números aleatorios). Como principio general los detalles no se explicitan hasta que no son necesarios con lo que el lector del programa puede concentrarse con mayor facilidad en su lectura y comprensión, lo que radicalmente rompe con las rígidas construcciones sintácticas habituales de los lenguajes de programación. El resultado es que el programa literario se lee con mayor naturalidad.

PROBLEMAS

La introducción de la programación literaria en un primer curso universitario de programación puede plantear una serie de dificultades importantes en la medida que el alumnado disponga de algunos conocimientos previos de programación por rudimentarios que sean. Estos conocimientos provienen del esfuerzo individual y de la lectura de revistas comerciales y libros de "bricolaje" informático. Este tipo de estudiantes, muy comunes en las escuelas de ingeniería, ingresan en la universidad con una serie de expectativas sobre lo que se les va a enseñar y tienen una fuerte preferencia hacia la enseñanza de los aspectos sintácticos del lenguaje de programación más en boga en ese momento (C hace unos diez años, luego C++, ahora Java, etc.) por lo que sin duda alguna no simpatizarán con el enfoque propugnado por la programación literaria. Además en el mundo laboral de la programación de ordenadores en la actualidad no se hace ningún tipo de hincapié en la documentación del software y desgraciadamente es una situación muy común el que no se requiera en absoluto. Por tanto aquellos estudiantes con experiencia laboral o que tengan contacto con programadores profesionales serán considerablemente escépticos acerca de las ventajas que pudiera aportarles una enseñanza literaria de la programación.

Muchos estudiantes creen firmemente en la "mística del programador solitario" que es aquel que trabaja aislado, sin haber recibido enseñanza formal y que demuestra su valía profesional realizando programas muy extensos y complejos, sin documentación y sin metodología alguna. Este tipo de programador era común en la industria hasta hace pocos años y su "leyenda" está firmemente asentada. Los aspirantes a "programadores solitarios" quedarán desagradablemente sorprendidos por el énfasis que la programación literaria realiza sobre la documentación del software y por el tiempo y disciplina de pensamiento que requiere la redacción de un buen programa literario.

Aunque resulte paradójico, el campo de las tecnologías de la información es mucho más conservador en su práctica profesional de lo que pudiera intuirse y resulta ser muy poco permeable a las ideas provenientes de la comunidad universitaria. Es una actitud muy común y fácilmente contrastable el no desear aprender nada nuevo si ya se tiene algún conocimiento en el área. Lo general es el deseo de obtener resultados rápidos utilizando esencialmente los mismos métodos ya conocidos y que todos usan. Quizás esta actitud sea un importante factor explicativo de la salida laboral como programadores de muchos universitarios no formados en programación (matemáticos, físicos, etc.)

Otros problemas de índole secundario son:

- La identificación de la programación literaria con el sistema operativo UNIX, resultado de su origen académico y que variará conforme se elaboren nuevas herramientas.

- El volumen de documentación a redactar por el profesor en un curso de programación literaria es muy superior al hoy día utilizado en la enseñanza de la programación. Sin embargo se tiene garantizada la publicación de unos "apuntes de clase" de calidad.
- Los alumnos ingresan en la universidad con una pobre capacidad de expresión escrita.
- La evaluación de un programa literario puede ser una tarea delicada pues es difícil el puntuar la exposición y claridad de estilo de un texto.
- Por supuesto vuelve a surgir la vieja cuestión de qué hacer con las faltas de ortografía.

CONCLUSIONES

Existe una importante separación entre la enseñanza universitaria de la programación de ordenadores y su desarrollo profesional. Es la industria informática la que realmente guía el cambio en esta área; basta con observar la abundante oferta de formación privada orientada exclusivamente al mundo laboral y que goza de un enorme éxito (certificaciones de Microsoft, Oracle, etc.) a pesar de su alto coste económico. La enseñanza universitaria de la programación no debe renunciar a intentar influir en la industria mediante la formación de sus futuros empleados. La programación literaria propone una enseñanza con énfasis en la redacción de documentación, la legibilidad de los programas y su mantenibilidad. Fomenta el trabajo cooperativo entre los alumnos y les obliga a mejorar su capacidad de expresión escrita y a comprender, estructurar y explicitar mejor los programas que realicen. Estas habilidades les serán sin duda muy útiles en sucesivos cursos universitarios y potenciarán su futuro desempeño profesional.

BIBLIOGRAFÍA

[1] BAECKER, R.M. y MARCUS, A. *Human Factors and Typography for More Readable Programs.* ACM Press/Addison-Wesley, 1990.

[2] DEIMEL, L.E. y NAVEDA, F. *Reading Computer Programs: Instructor's Guide and Exercises.* Carnegie Mellon University, 1990. Véase la página Web *Code Reading and Program Comprehension Bibliography* en el siguiente URL: http://www2.umassd.edu/SWPI/ProcessBibliography/bib-codereading.html

[3] FRASER, C.W. y HANSON, D.R. *A Retargetable C Compiler: Design and Implementation.* Addison-Wesley, 1995.

[4] HANSON, D.R. *C Interfaces and Implementations: Techniques for Creating Reusable Software.* Addison-Wesley, 1996.

[5] HOLUB, A.I. *Compiler Design in C.* Prentice Hall, 1990.

[6] KERNIGHAN, B.W. y RITCHIE, D.M. *El lenguaje de programación C, 2a. ed.* Prentice Hall, 1991.

[7] KNUTH, D.E. *TeX: The Program.* Vol. B of *Computers and Typesetting.* Addison-Wesley, 1986.

[8] KNUTH, D.E. *Metafont: The Program.* Vol. D of *Computers and Typesetting.* Addison-Wesley, 1986.

[9] KNUTH, D.E. *Literate Programming.* Center for the Study of Language and Information (CSLI), Lecture Notes No. 27, Stanford University, 1992.

[10] KNUTH, D.E. *The Stanford Graphbase. A Platform for Combinatorial Computing.* ACM Press/Addison-Wesley, 1993.

[11] LAMMERS, S. *Programadores en acción*. Anaya Multimedia, 1988.
[12] TANENBAUM, A.S. *Sistemas operativos: diseño e implementación*. Prentice Hall, 1988.
[13] TONDO, C.L. y GIMPEL, S.E. *The C Answer Book*. Prentice Hall, 1989.
[14] WIRTH, N. *Algoritmos + estructuras de datos = programas*. Eds. del Castillo, 1980.