Un modo de entender la programación

Rosana Satorre, Patricia Compañ, Faraón Llorens
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universidad de Alicante
03080 Alicante
e-mail: {rosana, patricia, faraon}@dccia.ua.es

Resumen

En este trabajo tratamos de aportar nuestra visión personal de cómo se debe iniciar a un alumno en el campo de la programación. No nos centramos en los contenidos a tratar, que vienen determinados por el plan de estudios, sino en aspectos más bien metodológicos, comentando el enfoque que le damos a la asignatura Fundamentos de Programación.

1. Introducción

Posiblemente el título de este artículo nos lleve a dudar sobre la adecuación al tipo de congreso presentado. Pero sí, estamos convencidos de que éste es el entorno de debate adecuado.

Al hablar de un modo de entender, nos referimos al modo de ver, de enseñar, de aprender la programación, un modo diferente según el entorno y el grupo del que formemos parte. Por ejemplo, un estudiante no ve la programación como la puede ver un profesor de Fundamentos de Programación, ni éste a su vez, la verá igual que un profesor de Sistemas Operativos, y así podríamos seguir, hasta encontrar posiblemente tantos puntos de vista y enfoques distintos como áreas de interés en programación. Por ello, en este artículo queremos mostrar el punto de vista de un profesor de Fundamentos de Programación, que como veréis os puede resultar un tanto peculiar.

A la hora de impartir una asignatura de programación y mucho más si es la primera con la que el estudiante se encuentra en su titulación, te haces muchas preguntas ¿cómo enseño?, ¿por dónde empiezo?, ¿qué paradigma utilizo?, ¿qué lenguaje de programación empleo?, ¿cómo oriento los ejercicios?, y lo peor ¿cómo imparto las clases teóricas? … Acerca del paradigma de programación con el cual empezar se ha debatido reiteradamente en ediciones anteriores del Jenui [1,2]. Nosotros no vamos a entrar en ello, pues independientemente del paradigma tendrán que escribir algoritmos que resuelvan determinados problemas. Para todas ellas en algún momento encuentras respuestas que en ocasiones sirven, soluciones que funcionan muy bien en algunos grupos y no tanto en otros. Así que año tras año te vuelves a plantear esas mismas preguntas y vuelves a encontrar otras respuestas, y de nuevo, vuelta a empezar, aunque no del todo, pues la experiencia es un grado.

2. Qué lenguaje empleamos

Puede parecer una pregunta con fácil respuesta: el lenguaje con el que yo aprendí, el de siempre, Pascal, el que dan todos. Posiblemente si lo enseñan todos es porque se trata del lenguaje adecuado. Pero, aquí nos encontramos nosotros, que no estamos convencidos del todo, que queremos mejores resultados, que esperamos ofrecer algo más de lo que recibimos como estudiantes, no porque no fuese lo adecuado, sino porque pensamos que todo, incluso nuestras propuestas siempre son mejorables. Así que a ello, a pensar qué lenguaje puede ser el más apropiado.

Nos planteamos emplear el lenguaje C, en principio estaría bien, ¿por qué no?, además en las asignaturas de los cursos siguientes les puede venir muy bien pues es el más utilizado. Pero al ponerlo en práctica durante varios cursos, vemos con gran decepción, que los estudiantes aprenden como normas de programación sus defectos, sus *bugs*, sus particularidades. Les intentas explicar que esto o aquello se debe escribir en el programa para resolver un pequeño conflicto y ellos se quedan con que hay que ponerlo siempre, se trate de un programa en lenguaje C o no. Ya habíamos oído y leído que el lenguaje C no es el idóneo para enseñar a programar a un estudiante que se inicia en estas labores [3].

Entonces, nos planteamos el lenguaje Pascal, muy bien, según muchísimos entendidos se trata

de un lenguaje perfecto para docencia. Y sí, lo es, pero al tratarse de un lenguaje completo, permite que el alumno se centre en aspectos que no interesan a la hora de resolver un problema por mucho que insistas en ellos, se preocupan mucho más de las florituras que de conseguir que un bucle funcione correctamente, por mencionar un caso concreto. Además al dárselo todo hecho, por poner un ejemplo, las cadenas, no se plantean, por qué funcionan, no entran en el detalle de cómo se consigue recogerla de una vez y almacenarla en un vector (cadena). Sí, tú debes insistirles, pero no importa, lo que cuenta es lo que ven y lo que hacen y de las cadenas no se deben preocupar, funcionan porque funcionan, porque es así, cosa de magia.

Y de nuevo, tras algunos cursos empleando Pascal, te vuelves a plantear qué lenguaje utilizar. Nosotros, los profesores de Fundamentos de Programación, no queremos que el programa sea perfecto, con animación, ventanas, sonidos, etc., eso sí, queremos que tenga la estructura perfecta, el resto, las florituras no aportan nada al algoritmo, sólo hacen bonito y eso, pensamos nosotros, entra en otra fase.

Por otro lado, nuestros estudiantes, con la llegada de las interfaces gráficas, han perdido los conocimientos que, sin darnos cuenta tuvimos que adquirir los usuarios del "viejo" ms-dos, ¿qué son las carpetas?, ¿por qué cuando un programa se atascaba y nos dedicábamos a pulsar teclas sin cesar, cuando por fin se recuperaba del error, empezaba a ejecutarse la secuencia de teclas que habíamos pulsado? (buffer de teclado).

Así que no queremos que les ocurra lo mismo al aprender a programar empleando un lenguaje de programación, queremos que sepan cómo funcionan realmente las cadenas, y por qué hay que liberar el buffer, o por qué debemos controlar el final de un vector, …. Y pensamos que ese conocimiento se adquiere mejor utilizando un lenguaje algorítmico con una sintaxis muy limitada, dónde se van a tener que plantear muchas cuestiones para conseguir lo mismo que si utilizasen un lenguaje de programación.

3. Cómo orientamos nuestras prácticas

Una vez decidido el lenguaje de programación a emplear, en nuestro caso, un lenguaje algorítmico con una sintaxis específica diseñada por nosotros mismos y muy restringida, nos encontramos ante otra dificultad, nuestros estudiantes son reacios a trabajar única y exclusivamente en papel. Otra barrera a franquear.

Con la intención de que las prácticas sean más atractivas se les plantean ejercicios resueltos, ejercicios propuestos breves, juegos, ejercicios similares a casos concretos reales, etc.

Mentalmente deben ser capaces de estructurar la resolución al problema. Al contrario de lo que ellos opinan, nuestro objetivo como profesores de Fundamentos de Programación no es que los programas o algoritmos, en este caso, estén perfectamente escritos, es decir, con los puntos y coma situados correctamente, con la palabra "si" en lugar de "if" si se trata de una lenguaje algorítmico en castellano, etc.. Nuestro objetivo es que ellos consigan estructurar perfectamente la solución del problema. No es tan importante al principio ver si una u otra variable vale uno u otro valor, sino componer la solución, estructurar el modo de resolver el problema.

Con esta finalidad nos sacamos de la manga los "diagramas de flujo mudos" y los "diagramas de cajas mudos". Queremos que de un modo gráfico y sin contenido nos indiquen cómo resolver un determinado problema.

Para comprender la idea veamos un ejemplo: rellenar de números consecutivos y en espiral, empezando por el 1, una matriz de dimensión $n \times m$ empleando un diagrama de cajas mudo.

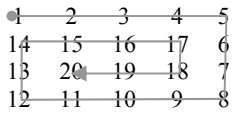

Figura 1. Ejemplo a resolver

En la solución queremos reflejar que para resolver el problema hacen falta cuatro estructuras de repetición, una para cada sentido de la espiral (de izquierda a derecha, de arriba abajo, de derecha a izquierda y de abajo a arriba) dentro de otra estructura de repetición que permitirá repetir el ciclo hasta completar la matriz. Empleando un

diagrama de cajas mudo quedaría del siguiente modo:

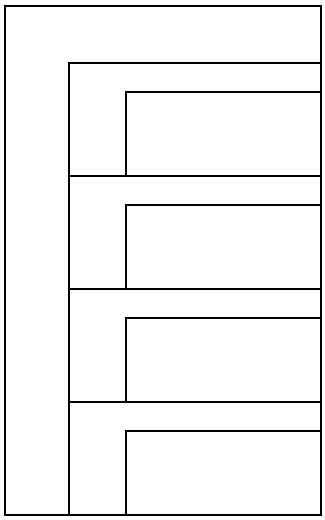

Figura 2. Diagrama de cajas mudo que resuelve el ejemplo de la ilustración 1

Evidentemente la solución final en forma de programa estará llena de matices, pero si un estudiante es capaz de diseñar mentalmente la estructura anterior tendrá más de la mitad del camino hacia la solución final resuelto.

Sin embargo, lo sorprendente es que cuando se les pide esto, ellos escriben primero la solución en lenguaje algorítmico para posteriormente dibujar el diagrama mudo y por mucho que lo intentamos no hay modo de modificar este hábito, sobre todo en aquellos que cursan por segunda vez la asignatura.

Y es que les cuesta ver el problema en su conjunto, de forma general, sin el detalle. Ellos le dan más importancia a si algo termina en punto y coma o no, como si eso fuera lo trascendental. Para ellos tiene tanta importancia que los días previos al examen reflejan su preocupación en preguntas del tipo: ¿Si no se pone ";" al final de una instrucción cuanta nota quita?, ¿si en lugar de escribir la asignación con "←" la escribimos ":=" descuenta mucho?

Se esmeran en que las cosas funcionen, a cualquier precio, con estructuras incorrectas e incoherentes, con el número de parches que haga falta, pero que funcione, aunque falle en ocasiones, pues si esto ocurre incorporan un nuevo parche y a funcionar.

Para intentar que comprendan que lo importante en un algoritmo no es el hecho de que funcione (esta característica debe ser aplicable a un programa), sino que tenga una estructura correcta, un paso de parámetros adecuado, un buen empleo de las variables, pues ello llevará al funcionamiento final, solemos escribir en la pizarra

"La bentana hesta havierta"

Al leer esta frase entienden que sí, que aunque se consiga el cometido, es decir, transmita el mensaje de que la ventana está abierta, la frase no es ortográficamente correcta y si se tratase de una asignatura de lengua no se admitiría. Eso mismo pretendemos que hagan ellos con sus algoritmos, que construyan algoritmos correctos. Haciendo un símil, la gramática en lengua es como la estructura en un programa, tanto a nivel de algoritmo como a nivel de datos.

Como práctica, les invitamos a leer el artículo de Agustín Cernuda "Cómo no hacer una práctica de programación" [4] y efectivamente, tal y como comenta Agustín les entusiasma y se ven reflejados en él, aunque no hemos podido constatar si repercute de manera positiva en sus ejercicios, pues la leen prácticamente al final del cuatrimestre, y por tanto al final de la asignatura.

Otro tipo de práctica que nos parece muy interesante incluir es un ejercicio de trabajo en grupo, pero no en el sentido de formar un grupo de trabajo y que hagan un ejercicio entre todos los componentes del grupo repartiéndose tareas. La idea es que el profesor reparta a diversos grupos de trabajo módulos que forman parte de un gran problema, indicando a cada grupo el objetivo del módulo, así como los parámetros de entrada, de salida y de entrada/salida que debe tener, sin que tengan constancia del enunciado del problema que se pretende resolver, únicamente de la parte que a ellos les concierne. Después se juntan todos los módulos para que vean que si no cumplen exactamente las especificaciones que se les han dado, el programa completo no funcionará.

En este tipo de prácticas ellos descubren la importancia de cumplir las especificaciones, pero no antes de empezar a resolver el ejercicio, pues

van directamente a escribir código y se olvidan de ello. Es después, cuando se construye el programa final y ven que no funciona, cuando aprenden que no seguir de manera rigurosa las instrucciones, les lleva a no poder enlazar su módulo con el resto.

La realidad es que cuando les pides a los alumnos que lean con detalle los enunciados parece que les solicites algo extraordinario: se limitan a echarle un vistazo rápido, y corriendo a escribir código que es lo realmente importante a su modo de ver, sin análisis ni diseño ni nada.

Otro tema en el que procuramos insistir, no con mucho éxito la verdad, es la documentación. Nuestros alumnos detestan escribir comentarios y no digamos elaborar la documentación externa, para ellos lo importante es el código. Sólo escriben la documentación al final, y eso porque se la exigimos como parte del ejercicio. Tratamos de inculcarles que si en un futuro trabajan en una empresa como programadores profesionales, es muy habitual que a una persona le toque ampliar o revisar el código que ha realizado otro programador, que a lo mejor ya ni siquiera pertenece a la empresa. Si el programa está bien documentado, le será más sencilla la tarea. Por bien documentado no entendemos sólo que tenga una buena explicación de lo que hacen los distintos módulos que forman el programa, es importante poner comentarios en los "puntos oscuros" o conflictivos, entendiendo como tales aquellas líneas de programa que pasado un tiempo es fácil no acordarse de por qué las pusimos. Tampoco entienden que la documentación externa va ligada al programa, que si actualizan el fuente, deben actualizar la documentación, en caso contrario no sirve para nada.

Apoyándonos en la cita de Match Kapor: *"Los programadores trabajan en la forma en que los artesanos medievales construían catedrales: piedra por piedra",* intentamos enseñarles que a la hora de diseñar un algoritmo para resolver un problema es muy importante el orden en el que se organicen las instrucciones. Les proponemos como trabajo ejercicios del tipo: "Analiza el problema de comprar una lata de refresco en una máquina de venta automática y diseña un algoritmo para resolverlo expresándolo en lenguaje castellano".

```
inicio
  tomar una moneda
  si queda el refresco elegido
    repetir
          introducir la moneda en
          el agujero preparado
          para ello
    hasta que no devuelva la
    moneda
    pulsar el botón del refresco
    seleccionado
    mientras salgan monedas
          recoger la moneda
          recoger la lata de
          refresco
  si no queda refresco guardar la
    moneda
fin
```

Siguiendo con el ejemplo de los maestros, Platón decía: "*Ninguna enseñanza obligatoria puede permanecer en el alma … Al educar a los niños, enséñales como si fuera una especie de juego y podrás ver con más claridad las inclinaciones naturales de cada uno*". Así pues, otro modelo de práctica es el aprendizaje mediante juegos, pues están tan interesados en jugar y en que sus compañeros jueguen que esto les motiva para no sólo hacer bien el ejercicio, sino terminarlo.

Como ejemplo de este tipo de prácticas proponemos un test de inteligencia. En dicho ejercicio deben emplear tablas y compararlas para dar el resultado final. Se trata de algo muy simple, sin embargo, si en lugar de emplear este problema, les planteáramos algo más teórico tardarían más en resolverlo, no porque le tuviesen que dedicar más tiempo, sino porque nunca encontrarían el momento de sentarse y ponerse a trabajar, porque no les motiva.

4. Cómo vivimos las clases teóricas

Las clases teóricas, tal y como se interpreta habitualmente la palabra teórica, no tienen mucho sentido en la asignatura Fundamentos de Programación, pues ¿cómo explicas teóricamente un algoritmo?, ¿mediante un resumen?. Hay un antiguo proverbio chino que dice "*Escucho y olvido. Veo y recuerdo. Hago y comprendo."* Nosotros lo llevamos todavía más allá, pues consideramos que si una persona lo tiene que explicar a otra lo comprenderá mejor, pues debe transmitir ese conocimiento, pero si además tiene que programarlo mediante un ordenador, su nivel

de comprensión tendrá que ser todavía mayor, pues las posibilidades de entendimiento de una máquina son nulas [5].

Y sobre todo, les inculcamos que la práctica hace maestros, que más vale dedicar dos horas a analizar un problema y resolverlo de manera incorrecta, que escuchar diez minutos al profesor y "aceptar" su perfecta solución al problema.

Esto nos lleva a pensar en clases más participativas, con numerosos ejercicios, con "trucos", con comentarios sobre costumbres y hábitos de programación, con comparaciones con situaciones reales, nada que ver con la teoría de libro.

Un buen método para que reflexionen sobre cualquier estructura o tipo de datos, es colocar frente a ellos la sintaxis de dicha estructura y esperar que la interpreten, que intenten entender cómo funciona y para qué, para posteriormente explicar cómo y dónde se utiliza.

Otro método puede ser el de plantear un ejercicio en el que seguro necesitarán una estructura o tipo de datos que no han visto todavía y ver cómo resuelven esa carencia.

También nos planteamos utilizar algún tipo de sistema de visualización de programas, algo parecido a Jeliot [6]. Este sistema utiliza animaciones para enseñar las estructuras de programación. Normalmente se entiende que una representación gráfica de un programa, y más si es con animaciones, siempre es mejor que el código fuente escrito en texto. Esto depende de la percepción personal: hay quién piensa que una imagen vale más que mil palabras, pero no siempre sucede lo mismo. Pongamos un ejemplo: Lees un libro y te gusta mucho, un tiempo después estrenan una película en el cine basada en dicho libro, vas corriendo a verla y sufres la gran decepción, ¿qué ha sucedido?.

5. Conclusiones

Desde nuestro punto de vista, enseñar programación no es algo sencillo, no se trata de ir enumerando las distintas estructuras indicando cada una de ellas para qué sirve, cómo se utiliza, que variables contiene, … . Es mucho más que eso, nuestro objetivo es que aprendan a pensar, a analizar una situación y a diseñar el método de resolución más adecuado, dejando al margen el lenguaje de programación.

En principio este objetivo puede parecer sencillo, pero la experiencia nos dice que no es así. Los alumnos se centran mucho más en detalles irrelevantes, en partes concretas del problema, en el resultado final, en lo que se muestra por pantalla, en florituras, etc., ignorando y olvidando el contexto general del problema.

La adquisición de la capacidad para escribir programas correctos, eficientes, bien organizados y adecuadamente documentados es un requisito esencial para cualquier titulado en informática, porque programar es algo más que conocer un determinado lenguaje.

Referencias

[1] García Molina, J. *¿Es conveniente la orientación a objetos en un primer curso de programación?* JENUI 2001 - VII Jornadas de Enseñanza Universitaria de la Informática.

[2] Gómez, A., González, J. y Lozano, A. *Tres lenguajes distintos y un solo objeto verdadero: una propuesta de introducción a la programación*. JENUI 2001 - VII Jornadas de Enseñanza Universitaria de la Informática.

[3] Llorens Largo, F. *Proyecto Docente: Lógica de Primer Orden, Ampliación de Lógica y Fundamentos de Programación*. Cuerpo de Profesores Titulares de Escuela Universitaria. Alicante 1996.

[4] Cernuda del Río, A. *"Cómo NO hacer unas prácticas de programación"* JENUI 2002 - VIII Jornadas de Enseñanza Universitaria de la Informática.

[5] Gal-Ezer, J & Harel, D. *What (Else) Should CS Educators Know?* Comunications of the ACM, Vol 41, nº 9, 1998

[6] Ben-Ari, M. "*La visualización de programas en la teoría y en la práctica*". Novática nº 150 2001