Actas de las XIX Jenui. Castellón, 10-12 de julio 2013
ISBN: 978-84-695-8051-6 DOI: 10.6035/e-TIiT.2013.13
Páginas: 225-232

Identificando algunas causas del *fracaso en el aprendizaje de la recursividad*: Análisis experimental en las asignaturas de programación

Carmen Lacave, Ana Isabel Molina, Juan Giralt
Departamento de Tecnologías y Sistemas de Información
Universidad de Castilla-La Mancha
Paseo de la Universidad, 4
13071 Ciudad Real
{Carmen.Lacave, AnaIsabel.Molina, Juan.Giralt}@uclm.es

Resumen

La recursividad es una herramienta muy potente para la solución de problemas complejos, sin embargo constituye uno de los conceptos más difíciles de entender por los alumnos cuando están aprendiendo a programar. En este artículo se describe una experiencia desarrollada en las asignaturas de *Fundamentos de Programación I* y de *Metodología de la Programación* en la Escuela Superior de Informática en Ciudad Real, que tenía como objetivo identificar las necesidades del alumnado a la hora de enfrentarse a la asimilación del concepto de recursividad. El hecho de haber realizado la experiencia en distintos cursos nos ha servido para identificar empíricamente los aspectos que más dificultades les suponen en distintas etapas de su aprendizaje. El estudio que se presenta en este artículo nos ha permitido contrastar la opinión y experiencia de los distintos grupos de estudiantes. Las conclusiones de esta experiencia y las lecciones aprendidas permitirán diseñar en el futuro una herramienta para la visualización de la recursividad que se adapte a las distintas necesidades del alumno, dependiendo de la etapa de aprendizaje en la que se encuentre.

Abstract

Recursion is a powerful tool for solving complex problems but it really is one of the most difficult concepts for students to understand when learning to program. This article describes the experience developed in the subjects of Fundamentals of Programming I and Programming Methodology in the School of Informatics in Ciudad Real in order to identify the needs of students when faced with the assimilation of recursion. The fact that the experience has been carried out in different courses has allowed us to identify empirically those aspects more difficult for the understanding of recursive algorithms at different stages of the learning. The study presented in this article allows us to contrast the opinion and experience of different groups of students. The conclusions of this experience and the lessons learned will enable design a tool for visualization of recursion to suit different needs depending on the student's learning stage.

Palabras clave

Experiencia docente, recursividad, programación, análisis de dificultades.

1. Introducción

La *recursividad* es una técnica de programación muy potente, puesto que permite resolver problemas muy complejos en función de las soluciones de los mismos problemas, pero de menor tamaño, basándose en el principio de inducción matemática. El poder de la recursión se basa en la posibilidad de definir un conjunto infinito de objetos o de operaciones computacionales con una declaración finita [11]. Es por ello que el concepto de recursividad aparece en todos los currículos de los primeros cursos de programación.

Sin embargo, la recursividad es uno de los principales obstáculos al que se tienen que enfrentar los alumnos cuando comienzan a programar, por lo que desde hace tiempo se trabaja intentando conocer las principales causas por las que resulta tan complicado dominarla [8,10], así como en proponer distintos métodos que ayuden a paliar estas dificultades [4,10]. Una de las tendencias actuales es utilizar *técnicas de visualización*, ya que a pesar de que no hay evidencias contundentes de que tenga como consecuencia una mejora en el proceso de aprendizaje [5], es un elemento motivador para el estudiante [3,9].

Los esfuerzos han ido encaminados al desarrollo de diferentes herramientas de visualización de programas recursivos [1,7] (véase Perez Carrasco para una revisión completa), pero no existen trabajos que hayan ofrecido datos empíricos que justifiquen el por qué del uso de unas u otras herramientas de visualización, o que justifiquen el empleo de una metodología específica. Además, las propuestas que se han realizado no inciden en las distintas necesidades del alumnado. De hecho, el mayor inconveniente de las herramientas de visualización más conocidas, y específicas para programas en Java, *Srec* [7] y *Jeliot*[1], es que no se puede determinar qué elementos de la recursividad son aquellos sobre los que interesa incidir, dependiendo de la etapa del aprendizaje en la que se encuentra el alumno. Por ejemplo, el programa *Srec* es demasiado complicado para los alumnos de primer curso puesto que aún no entienden el concepto de pila o de árbol, que son las estructuras visuales que esta herramienta utiliza para representar gráficamente el comportamiento de las llamadas recursivas. En el caso de *Jeliot,* la interfaz puede llegar a ser demasiado confusa para los alumnos debido a la cantidad de información que ofrece. Otros programas, como el propuesto en [2] tienen una interfaz que genera una ventana distinta para visualizar cada llamada recursiva, lo que puede resultar más intuitivo para los alumnos de 1º; sin embargo, no está disponible.

Puesto que el diseño de herramientas de visualización supone una alta carga de trabajo para quienes generan las visualizaciones (generalmente profesores) [5], conviene determinar previamente qué elementos de la recursividad son los que más trabajo les cuesta entender a los estudiantes en función del conocimiento previo adquirido, con el fin de elegir convenientemente las técnicas de visualización que mejor se adapten a sus necesidades.

Por tanto, nuestro *objetivo* es el de identificar los elementos o características de la recursividad en los que es necesario incidir más dependiendo de las necesidades de cada estudiante según la etapa del aprendizaje en la que se encuentre. No es lo mismo enfrentarse a la recursividad en primer curso, donde aún no tienen claro el concepto de pila y ni siquiera conocen el de árbol, que en segundo curso, en el que ya han estudiado el manejo de las estructuras de datos más complejas. Para ello, hemos realizado un estudio con los estudiantes matriculados en las asignaturas de *Fundamentos de Programación I* y de *Metodología de la Programación*, cuyos resultados constituyen la principal contribución de este trabajo.

En las siguientes secciones se introducen generalidades sobre la recursividad (Sec. 2) y su estudio dentro de las asignaturas básicas de programación en el Grado de Informática de la Escuela Superior de Informática de Ciudad Real (ESI) (Sec. 3). A continuación, se presentan los detalles del estudio realizado con los estudiantes matriculados (Sec. 4) y los resultados obtenidos (Sec. 5). Finalmente, se apuntan algunos comentarios finales y las lecciones aprendidas (Sec. 6).

2. El aprendizaje de la recursividad

La *recursividad* es la estrategia mediante la que se define un proceso en base a su propia definición. Esta técnica se basa en el principio de inducción matemática, que permite suponer resuelto el problema para todos los casos de menor tamaño que él. El aprendizaje de la recursividad implica el dominio de dos aspectos básicos y complementarios: el primero está relacionado con el *funcionamiento,* esto es, con los procesos que se llevan a cabo internamente para que la llamada recursiva produzca los resultados esperados, es decir, el **cómo** lo hace; el segundo, con el *diseño* de algoritmos recursivos, que se lleva a cabo de forma declarativa incidiendo exclusivamente en el **qué** hace.

Dentro del primer grupo, hemos identificado los elementos claves que se deben tener en cuenta para entender el *funcionamiento* de un algoritmo recursivo:

- *Caso base*. Se define la solución del problema para aquellos casos en los que es trivial o inmediata. Hay que asegurarse de que se llega siempre en un número finito de pasos.
- *Variables locales*. A pesar de que tienen el mismo nombre, son variables distintas e independientes en todas las llamadas, no se modifican los valores de unas llamadas a otras.
- *Paso de parámetros*: en el paso por valor, actúan de la misma manera que las variables locales; en el paso por variable o por referencia, actúan como variables globales, por lo que se puede modificar el valor de una llamada a otra.
- *Paso de objetos como parámetros:* actúan como las variables globales, siempre y cuando no haya operaciones de creación de instancias dentro de la propia llamada recursiva.
- *Vuelta* atrás. Se recuperan los valores de los parámetros por valor y de las variables locales y se devuelve el flujo de ejecución del programa al punto siguiente al de la llamada recursiva.

Un elemento clave propio de la recursividad es la gestión de las distintas *llamadas activas*, esto es, las llamadas que aún están ejecutándose. La mayoría de los lenguajes de programación utilizan pilas, por lo que conviene haberlas estudiado previamente. En el caso de llamadas recursivas múltiples, el funcionamiento se puede ilustrar mediante un árbol o un grafo, por lo que antes de introducir esta parte de la recursividad es aconsejable haber estudiado dichas

estructuras.

Respecto al *diseño* de los algoritmos recursivos, éste está íntimamente relacionado con el proceso de resolución de problemas de forma declarativa. Por tanto, no podemos enumerar una serie de elementos a tener en cuenta en el aprendizaje de esta etapa, sino que hacemos hincapié en el modo de modificar su forma de pensar. Para ello, proponemos dos etapas para diseñar con éxito un algoritmo recursivo:

- En la primera, se establecen los elementos que definen el tamaño del problema y se concretan uno o más casos bases y sus correspondientes soluciones.
- En la segunda, los alumnos deben partir de la idea de que el problema está resuelto ya para datos más pequeños, sin pensar en cómo se ha hecho, haciendo uso precisamente de su capacidad de abstracción para ignorar el desarrollo de todo el proceso. A partir de ahí, se busca la solución del problema en función de la solución que la recursividad, por "arte de magia", nos ha proporcionado.

3. La recursividad en las asignaturas de programación de la ESI

El concepto de recursividad aparece en todos los currículos de Informática, tanto en los de España como fuera de ella, normalmente entre los descriptores de las asignaturas de introducción a la programación de primer curso (CS1). En los estudios de Grado en Informática de la Escuela Superior de Informática, el concepto de la recursividad se imparte por primera vez en la asignatura de *Fundamentos de Programación I (FPI)*, al final del primer cuatrimestre, después de haber introducido los elementos básicos de programación, los principios de la programación estructurada y modular, y las estructuras de datos básicas. En la asignatura de *Fundamentos de Programación II (FPII)*, sin embargo, no se hace demasiado hincapié en trabajar la recursividad puesto que en esta asignatura se presentan los principios de la programación orientada a objetos, el tratamiento de excepciones y la programación orientada a eventos. Es en segundo curso cuando se vuelve a presentar la recursividad como tema transversal en la asignatura de *Estructuras de Datos (ED)*, que se imparte en el primer cuatrimestre del curso. El contenido de esta asignatura se dedica al estudio de las principales estructuras de datos lineales (pilas, colas y listas), y no lineales (árboles y grafos). Sin embargo, la recursividad es una herramienta indispensable en la asignatura de *Metodología de la Programación (MP)*, que se desarrolla en el segundo cuatrimestre de segundo curso, puesto que lo que se estudia en ella son algunas de las técnicas de programación meramente recursivas, como "divide y vencerás", *backtracking* y programación dinámica. Tanto es así que la principal causa que esgrimen la mayoría de los alumnos que abandonan la asignatura de *Metodología de la Programación*, o que no la superan, es que les falta dominio de la recursividad, por lo que se les hace muy difícil asimilar las distintas estrategias de resolución de problemas, lo que supone el abandono de la asignatura de un elevado número de alumnos desde las primeras clases. El problema es un poco contradictorio, pero sencillo de explicar. Cuando cursan esta asignatura ya han pasado dos cuatrimestres completos desde que aprendieron los primeros conceptos sobre recursividad y prácticamente no la han vuelto a ver hasta que llegan a *MP*. Esto es porque en *FPII* se dedican en exclusividad al aprendizaje de un nuevo paradigma de programación, el orientado a objetos. En la asignatura *ED* el estudio que se hace de las estructuras de datos es básicamente iterativo (a pesar de que de todas ellas, salvo los grafos, se pueden dar definiciones recursivas), dejando la recursividad para la resolución de algunos ejemplos aislados.

Por eso hemos decidido profundizar en el tema y analizar qué aspectos son los que más dificultades les suponen a los alumnos, con el objetivo de disminuir el fracaso en esta asignatura. Puesto que es difícil medir la capacidad de abstracción de los alumnos para detectar y corregir patrones incorrectos de pensamiento declarativo, nuestro trabajo lo hemos orientado a identificar qué elementos del funcionamiento de los algoritmos recursivos son en los que hay que profundizar, con el objeto de desarrollar, en una fase posterior, una herramienta que permita adaptarse a las distintas necesidades del alumno.

4. Las experiencias realizadas

El objeto de nuestro trabajo ha consistido en la realización de dos experiencias en las que se buscaba identificar las carencias y dificultades de los alumnos a la hora de aprender el concepto de la recursividad en distintas etapas de su aprendizaje.

4.1. Experiencia en la asignatura de *Fundamentos de Programación I*

La primera experiencia se realizó con 23 alumnos de primer curso de la asignatura FPI, que accedieron a realizar el experimento de forma voluntaria, y de los cuales 5 repetían la asignatura. La actividad se llevó a cabo en la última semana del cuatrimestre, después de haber visto el tema de recursividad y haber tenido tiempo para practicar con ella. A cada participante se

le citó de forma individual para rellenar un cuestionario donde tenían que aportar sus datos sobre edad, si era o no repetidor de la asignatura y valorar en una escala de 1 a 5 tres preguntas sobre su gusto por la programación *(MGP)* y su conocimiento sobre los principios teóricos *(CTR)* y el funcionamiento de la recursividad *(CFR)*. La Tabla 1 muestra la distribución de frecuencias de la primera parte del cuestionario. En ella podemos observar que a todos los alumnos les gusta bastante la programación. Por otro lado, la mayoría dice conocer los principios teóricos de la recursividad (65%) y el funcionamiento de la misma (70%).

N=23	1 (%)	2 (%)	3 (%)	4 (%)	5 (%)	>=3 (%)*
1. Me *gusta* programar ***(MGP)***	0 (0)	0 (0)	5 (22)	9 (39)	9 (39)	**23 (100)**
2. Conozco los *principios* teóricos de la recursividad ***(CTR)***	1 (4.3)	7 (30)	10 (43)	5 (22)	0 (0)	**15 (65)**
3. Conozco el *funcionamiento* de la recursividad ***(CFR)***	2 (8.7)	5 (22)	12 (52)	4 (17)	0 (0)	**16 (70)**
4. Valoración rep. 1: código solo ***(VALR1)***	1 (4)	5 (22)	9 (39)	5 (22)	3 (13)	**17 (74)**
5. Valoración rep. 2: tablas ***(VALR2)***	0 (0)	2 (9)	6 (26)	10 (43)	5 (22)	**21 (91)**
6. Valoración rep. 3: diagramas ***(VALR3)***	3 (13)	5 (22)	3 (13)	9 (39)	3 (13)	**15 (65)**

Tabla 1. Distribución de frecuencias de los datos de la experiencia 1 sobre un total de 23 cuestionarios. Las columnas corresponden a las valoraciones asignadas a cada una de las respuestas *La última columna representa el número de alumnos que han contestado a dicha pregunta con un valor mayor o igual a 3.

Además, cada alumno debía resolver individualmente tres ejercicios y valorar dos representaciones con las que se suele ilustrar el funcionamiento de los algoritmos recursivos de cara a facilitar la comprensión de la recursividad (Figura 1). Los dos primeros ejercicios consistían en proporcionar el resultado de la llamada a sendos métodos recursivos; en el tercero debían encontrar el error que aparecía en otro método recursivo. Los tres métodos estaban escritos en Java, puesto que es el lenguaje que aprenden en la asignatura de programación. Además, cada uno de los dos primeros ejercicios se ilustraba con una de las dos representaciones que se suelen utilizar en clase y que se muestran en la Figura 1: las tablas (Figura 1.A) y los diagramas (Figura 1.B), mediante los que se pretende representar el flujo de las distintas llamadas activas y la vuelta atrás de cada una de ellas. Para solucionar los tres ejercicios disponían de un tiempo máximo de 10 minutos. Los resultados de los ejercicios no son muy buenos: el primer ejercicio (Figura 1.A) lo resuelven correctamente 9 alumnos; 13 el segundo (Figura 1.B) y únicamente 5 encuentran el fallo del método recursivo. Por tanto, estos datos ratifican nuestra idea de que cuando están aprendiendo a programar, les cuesta mucho trabajo dominar la recursividad. Esta conclusión pudiera entrar en contradicción con los datos que aparecen en la Tabla 1, donde se pone de manifiesto que la mayoría de los alumnos creen que los principios teóricos de la recursividad y el funcionamiento de la misma.

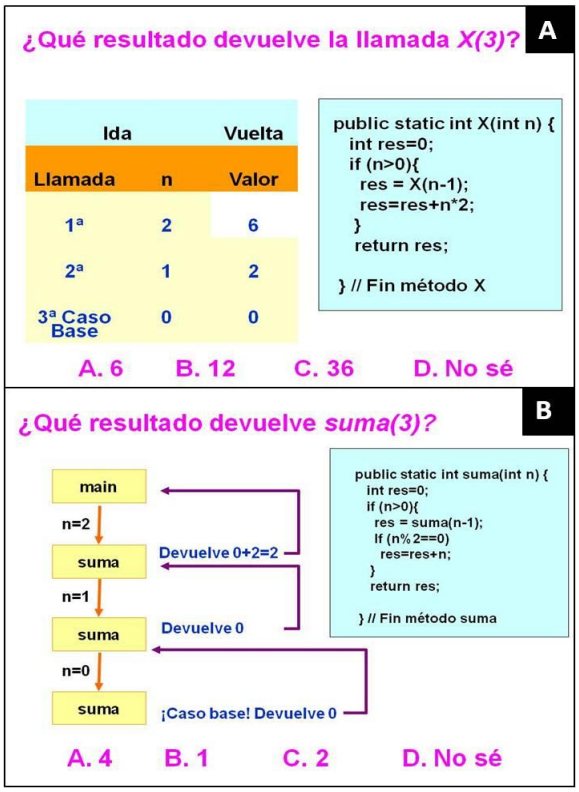

Figura 1. Dos de los ejercicios recursivos propuestos a los alumnos, ilustrado cada uno de ellos con un tipo de representación gráfica de la recursividad utilizado en la asignatura de FPI. 1. A: tablas; 1.B: diagramas.

Desde nuestro punto de vista esto no es así. Creemos que la recursividad es una estrategia lo suficientemente compleja como para que, a pesar de que se conozcan los principios sobre los que se fundamenta, los alumnos que empiezan a programar no son capaces de identificar fácilmente qué es lo que hace un algoritmo recursivo y, por tanto, qué resultados devuelve.

En cuanto al tipo de representación que mejor valoran, destaca la 2 (basada en diagramas). Esto coincide además con que el ejercicio que mejor resuelven también es el 2, lo que confirma nuestra hipótesis de que la visualización de la traza del algoritmo es importante para comprender el funcionamiento de la recursividad. En la Tabla 2 se resumen todos estos resultados, a través de los valores de los estadísticos más descriptivos: la media, desviación típica, mediana y moda.

	Edad	*MGP**	*CTR*	*CFR*	*VALR1*	*VALR2*	*VALR3*
μ	19,3	**4,17**	2,83	2,78	3,17	**3,78**	3,17
σ	3,27	0,78	0,83	0,85	1,07	0,9	1,3
Mediana	18	4	3	3	3	4	4
Moda	18	5	3	3	3	4	4

Tabla 2. Resumen de los datos correspondientes a la experiencia llevada a cabo con los alumnos de *FPI*. *Los nombres de las columnas 2 a 7 corresponden a las abreviaturas de los ítems 1 a 6 del cuestionario descritos en la Tabla 1.

4.2. Experiencia en la asignatura de *Metodología de la Programación*

En este caso, puesto que ya tienen mucha más experiencia en programación y, supuestamente, con la recursividad, pretendimos identificar los aspectos de esta técnica que más trabajo les cuesta entender, teniendo en cuenta el momento del aprendizaje en el que se encuentran.

Para ello, preparamos un cuestionario que respondieron voluntariamente 91 alumnos de los matriculados en la asignatura de *Metodología de la Programación* (31 de ellos repetidores de la asignatura). El cuestionario lo entregamos el primer día de curso de la asignatura en el 2º cuatrimestre y lo completaron al finalizar la clase. En el cuestionario debían valorar de 1 a 5 cada uno de los ítems subjetivamente. El cuestionario contenía 30 preguntas sobre aspectos relacionados con su conocimiento previo, su estilo de aprendizaje, su método de estudio, sus dificultades a la hora de entender y usar la recursividad y posibles formas de solucionarlas. Además, para contrastar los datos con los obtenidos en la experiencia de primero (descrita en la sección anterior) añadimos tres ejercicios que debían resolver, junto con la valoración de las representaciones de la recursividad que mejor les sirvieran para obtener el resultado correcto. Los tres ejercicios consistían en devolver el resultado de la llamada a sendos métodos recursivos escritos en Java. El primer ejercicio contenía simplemente el código del método; los dos últimos eran exactamente los mismos del experimento descrito en la sección anterior (Figura 1).

5. Resultados y Discusión

El análisis de los datos lo hemos realizado en distintas fases, que se describen en los apartados que siguen a continuación.

5.1 Considerando conjuntamente los datos comunes a los cuestionarios de 1º y 2º curso

Si analizamos los datos comunes a ambos cursos, resumidos en la Tabla 3, y los comparamos con los de la Tabla 2, podemos extraer algunas conclusiones interesantes.

	*MGP**	*CTR*	*CFR*	*VALR1*	*VALR2*	*VALR3*
μ	**4**	**3**	**3**	**2,8**	**3,4**	**3,4**
σ	1	1	1	1,1	1	1,2
Mediana	4	3	3	3	3	3,5
Moda	4	3	3	3	3	3

Tabla 3. Resumen de los datos correspondientes a la experiencia llevada a cabo con los alumnos de *MP*. *****Los nombres de las columnas 2 a 7 corresponden a las abreviaturas de los ítems 1 a 6 del cuestionario descritos en la Tabla 1.

La primera de ellas es que la media de *MGP* (ítem 1 del cuestionario) ha bajado ligeramente pero, sin embargo, las de *CTR* (ítem 2) y *CFR* (ítem 3) han subido, lo que es lógico pues han tenido otras asignaturas en las que han estudiado y utilizado la recursividad. Sin embargo, ahora se igualan las valoraciones de las representaciones.

Para intentar comprender el porqué de dichas variaciones, decidimos realizar un análisis de las correlaciones entre los datos, obteniendo los resultados que se muestran en la Figura 2. A partir de ellos podemos deducir que los que están en 2º curso tienen un mayor número de aciertos (0,35), aunque el hecho de que el valor de correlación no sea cercano al 1 nos hace pensar que en las demás asignaturas de programación que cursan durante el segundo cuatrimestre de 1º (*FPI*) y el primero de 2º (*ED*) no mejoran demasiado sus conocimientos en recursividad. Además, aquellos a los que les gusta programar hacen una mejor valoración (0,23) de la representación número 2 (Figura 2.A), pero el hecho de estar en cursos superiores hace que disminuya la valoración de la misma (-0,19). Como el número de

	Edad	Curso	MGP	CTR	CFR	VALR1	VALR2	VALR3	Nº CONT	Nº BIEN
Edad	1.00									
Curso	**0.33*****	1.00								
MGP	-0.14	-0.12	1.00							
CTR	**0.22***	0.17	0.08	1.00						
CFR	**0.23***	**0.29****	0.01	**0.69*****	1.00					
VALR1	-0.15	-0.15	-0.08	-0.06	-0.14	1.00				
VALR2	-0.07	**-0.19**	**0.23**	-0.06	-0.03	0.15	1.00			
VALR3	0.14	0.11	0.16	-0.03	0.00	-0.06	**0.37*****	1.00		
Nº CONT	-0.05	**-0.29****	-0.02	0.06	0.16	0.18	0.02	0.00	1.00	
Nº BIEN	0.08	**0.35*****	0.14	**0.24****	**0.39*****	-0.02	0.00	0.02	**0.38*****	1.00

*=p< .05 (el mínimo coeficiente de correlación significativa para un tamaño muestral de n= 114 es 0.184)
**=p< .01 (el mínimo coeficiente de correlación significativa para un tamaño muestral de n= 114 es 0.240)
***=p< .001 (el mínimo coeficiente de correlación significativa para un tamaño muestral de n= 114 es 0.304)

Figura 2. Correlaciones entre los datos de las tablas 1 y 2.

alumnos de 2º curso (91) es mucho mayor que los de 1º (23) esto puede explicar que al final las valoraciones de las representaciones 2 y 3 se igualen. En lo que todos coinciden es en que prefieren visualizar una representación de la traza en lugar de enfrentarse solo al código del algoritmo, como era de esperar.

5.2. Considerando sólo los alumnos de 2º curso

En este caso, el cuestionario incluía también un apartado relativo a sus hábitos de estudio y de qué forma pensaban que podrían mejorar sus resultados; las preguntas de esta parte se detallan en la Tabla 4.

Puntúa los siguientes aspectos de 1 a 5 según lo que tú consideres te **ayudaría a mejorar, e**n caso de que tengas problemas con la comprensión y el diseño de programas recursivos, *(1: "nada'; 5:"todo"')*
9. Más ejercicios *resueltos por el profesor* en las asignaturas de 1º
10. Más ejercicios *realizados por mí* durante las asignaturas de 1º
11. Mejor material docente
12. Visualización de la traza de los algoritmos recursivos realizadas por el profesor
13. Realización, *por mi parte*, de las trazas de los algoritmos recursivos
14. Una herramienta software de *visualización y simulación* de programas recursivos

Tabla 4. Extracto del cuestionario con las preguntas relativas a las posibles formas de mejorar su dominio de la recursividad.

Los datos indican, como muestra la Figura 3, que lo que más les ayudaría a mejorar su conocimiento sobre la recursividad sería la realización de las trazas de los algoritmos recursivos (P13). Después, lo que más valoran es la utilización de una herramienta software de visualización y simulación de programas recursivos (P14) y la realización de más ejercicios durante las asignaturas de primer curso (P10).

Además, el cuestionario también incluía varias preguntas, detalladas en la Tabla 5, sobre el grado de dificultad que les supone los principales aspectos de la recursividad.

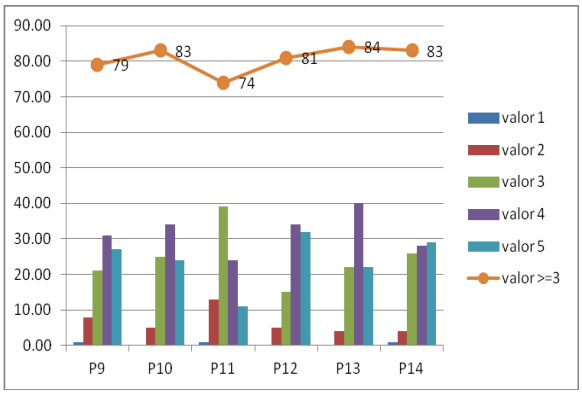

Figura 3. Resultados de las valoraciones de los alumnos a las preguntas de la Tabla 4. La serie naranja representa la suma de los alumnos que han asignado un valor mayor o igual a 3.

La Figura 4 describe los datos recopilados para las preguntas de la Tabla 5. Se observa que lo que más trabajo les supone al 84,6% de los alumnos encuestados es descifrar el funcionamiento del programa cuando existen varias llamadas recursivas (P21), seguido de la evolución de la traza de un algoritmo recursivo (P22) por el 81%, y la recuperación de los valores de variables y parámetros en la vuelta atrás de la llamada activa (P19) por el 77%. Podría parecer extraño que lo que menos dificultad les supone es el paso de objetos como parámetros (P20), pero es lógico pues actúan como si fueran variables globales.

Además, a pesar de las buenas percepciones subjetivas que tienen los alumnos sobre su conocimiento de la recursividad (Tabla 3), únicamente 29 de 91 han resuelto correctamente los tres ejercicios que se les plantearon. Este hecho sugiere que aprender a dominar la recursividad les

cuesta más trabajo de lo que ellos creen.

Valora de 1 a 5 cada una de las siguientes cuestiones según tu **grado de dificultad** para comprender la recursividad.
15. La llegada al caso base desde el caso general
16. Acceso y modificación de parámetros y variables locales
17. Acceso y modificación de las variables globales
18. Llamadas activas
19. Recuperación de los valores de variables y parámetros en la vuelta atrás de la llamada activa
20. Paso de objetos como parámetros
21. Funcionamiento del programa cuando existen varias llamadas recursivas
22. Seguimiento de la traza de un algoritmo recursivo

Tabla 5. Extracto del cuestionario con las preguntas relativas a los aspectos básicos de la recursividad.

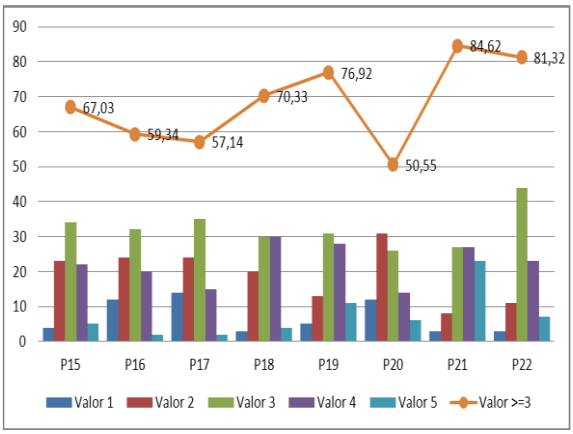

Figura 4. Resultados de las valoraciones de los alumnos a las preguntas de la Tabla 5. La serie naranja representa la suma de los alumnos que han asignado un valor mayor o igual a 3.

Asimismo, el análisis de correlaciones de los datos del cuestionario revela más información que merece la pena ser tenida en cuenta. Por falta de espacio, no incluimos una tabla con todos los resultados del análisis de correlaciones[1] y sólo destacamos los más relevantes:

- Los que más conocimiento dicen tener sobre los principios teóricos de programación son los que creen que conocen mejor el funcionamiento de la recursividad (0,67), la comprenden mejor (0,6) y valoran su utilidad (0,48). Por eso, son los que opinan que menos problemas tienen con el acceso y modificación de variables locales (-0,29), acceso y modificación de variables globales (-0,22), vuelta atrás en la recursividad (-0,26) y funcionamiento del programa con varias llamadas recursivas (-0,21).
- A aquellos que más trabajo les cuesta diseñar programas recursivos también les cuesta más trabajo comprender el funcionamiento del programa cuando existen varias llamadas recursivas (0,36) y son los que peor realizan el ejercicio 2 (-0,21).
- Lógicamente, a los que más trabajo les cuesta entender el acceso y modificación de parámetros y variables locales, también les cuesta el acceso y modificación de las variables globales (0,85), las llamadas activas (0,41), la vuelta atrás (0,41), el paso de objetos como parámetros (0,40) y el funcionamiento de un programa con varias llamadas recursivas (0,). Son también los que peor resuelven el ejercicio número 2 (-0,21).

La última fase del análisis ha consistido en aislar los datos de los 29 alumnos de 2º curso que han respondido bien a las tres preguntas y hemos procedido igualmente a realizar un análisis de correlaciones de los datos[2]. El único resultado destacable es que los que consideran que mejor conocen los principios y el funcionamiento de la recursividad son los que también consideran útil una herramienta software de visualización y simulación de programas recursivos (0,38).

6. Conclusiones y trabajo futuro

La recursividad es una herramienta muy importante y potente para la solución de problemas complejos. Sin embargo, aunque los alumnos creen que entienden sus principios teóricos y su funcionamiento, las experiencias llevadas a cabo con los alumnos que empiezan a programar en 1º, e incluso con los que llevan ya más de un año programando, en el caso de los de 2º, demuestran que no la dominan. Es más, les supone un gran obstáculo para superar asignaturas obligatorias como las de *Metodología de la Programación*, en la que el uso de la recursividad es clave para el desarrollo de las estrategias de resolución de problemas como "divide y vencerás, *backtracking* y programación dinámica. De hecho, los alumnos consideran que son precisamente sus dificultades a la hora de utilizar esta técnica una de las principales causas que les lleva a no superar la asignatura.

En este trabajo nos hemos planteado identificar, apoyándonos en datos empíricos, aquellos aspectos que más trabajo les supone dominar, con el objetivo de incidir más en ellos. Así, se concluye que los aspectos más problemáticos son los relacionados con

[1] El mínimo valor del coeficiente de correlación significativo para una muestra de n=91 es 0,21 para una confianza del 95%; de 0,27 para una del 99%, y de 0,34 para una del 999%.

[2] El mínimo valor del coeficiente de correlación significativo para una muestra de n=29 es 0,37 para una confianza del 95%; de 0,47 para una del 99% y de 0,58 para una del 999%.

la vuelta atrás, la recuperación de valores en las llamadas anteriores, así como el funcionamiento de la recursividad en los casos en los que existen múltiples llamadas recursivas.

La visualización de la traza les ayuda a la comprensión del funcionamiento y a resolver mejor los ejercicios, pero hay que definir herramientas adecuadas al nivel de aprendizaje de cada alumno, puesto que el nivel de conocimiento y, por tanto, los recursos del alumnado son diferentes en cada etapa de aprendizaje. Así, aunque todo el alumnado coincide en que es muy importante visualizar la traza del algoritmo, los recursos que se pueden utilizar en primer curso no pueden, ni deben, ser los mismos que los de segundo, donde ya se pueden utilizar pilas y árboles sin problemas.

Además, los resultados obtenidos al comienzo de este curso nos han servido para introducir en el temario de la asignatura de *Metodología de la Programación* de 2º un tema adicional dedicado exclusivamente a profundizar en la recursividad: tanto en los principios del diseño, como en el análisis del funcionamiento de la misma. Todo esto ya ha servido, de momento, para aumentar el interés del alumnado y disminuir el abandono prematuro de la asignatura, tal y como venía ocurriendo en años anteriores.

A lo largo del curso pretendemos realizar experiencias similares mediante las que vayamos midiendo cómo el hecho de incidir en los aspectos que más dificultades les supone repercute en su motivación y, por tanto, en su rendimiento académico.

Como trabajo futuro, tal y como se ha comentado anteriormente, nos planteamos la implementación de una nueva herramienta de visualización de la recursividad que permita adaptar el aprendizaje del alumno dependiendo de sus necesidades. Dicha herramienta está ya en sus primeras fases de desarrollo.

Referencias

[1] Ben-Bassat, R., Ben-Ari, M., Uronen, P.A. (2003). The Jeliot 2000 program animation system. Computers & Education, 40(1), pp.1-15

[2] Dershem, H.L., Erin Parker, D., Weinhold, R. 1999): A Java function visualizer. Journal of Computing in Small Colleges, Vol. 15.

[3] Hundhausen, C., Douglas, S., Stasko., I (2002), A Meta-Study of algorithm visualization effectiveness, Journal of Visual Languages and Computing, 13, 259-290.

[4] Levy, D., Lapidot, T. (2000), Recursively speaking: analyzing students' discourse of recursive phenomena ACM SIGCSE Bulletin, Vol 32(1), 315-319

[5] Naps, T., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C.,Korhonen, A., Malmi, L., McNally, M., Rodger, S., Velázquez, J. Á. (2003a): Exploring the Role of Visualization and Engagement in Computer Science Education. *ACM SIGCSE Bulletin*, Vol. 35(2), pp. 131-152.

[6] Naps, T., Cooper, S., Koldehofe, B., Leska, C., Rößling, G., Dann, W., Korhonen, A., Malmi, L., Rantakokko, J., Ross, R., Anderson, J., Fleischer, R., Kuittinen, M., McNally, M. (2003b): Evaluating the educational impact of visualization. *ACM SIGCSE Bulletin*, Vol. 35(4), pp. 124-136.

[7] Perez Carrasco, A., Sistema Generador de Animaciones Interactivas para la Docencia de Algoritmos Recursivos, Tesis Doctoral, URJC, 2011.

[8] Sooriamurthi, R., Problems in Comprehending Recursion and Suggested Solutions, Proceedings of the 6th annual conference on Innovation and Technology in Computer Science Education (ITiCSE), Canterbury, UK, June 2001, 25 - 28.

[9] Urquiza, J., Velázquez, J. Á. (2009): A Survey of Successful Evaluations of Program Visualization and Algorithm Animation Systems. *ACM Transactions on Computing Education (TOCE)*, Vol. 9(2), ACM Press: New York, NY, Article 9.

[10] Velázquez Iturbide, J.A. (2000), Recursion in gradual steps (is recursion really that difficult?) ACM SIGCSE Bulletin, Vol 32(1), 310-314

[11] Wirth, N. (1976) Algorithms+Data Structures=Programs. Prentice Hall.