Actas de las XX JENUI. Oviedo, 9-11 de julio 2014
ISBN: 978-84-697-0774-6
Páginas: 159-166

Evaluación de una innovación docente a través de un diseño estadístico cuasi-experimental: aplicación al aprendizaje de la recursividad

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

Resumen

Una de las principales dificultades que tienen que superar los alumnos que empiezan a programar es el aprendizaje de la *recursividad*. Nuestro interés se centra en conocer las causas que están detrás de esta problemática e intentar solventarla. Creemos que uno de los principales motivos que podrían explicar este fenómeno es el enorme desajuste conceptual que les supone el cambio de paradigma, de imperativo a declarativo. Con el objetivo de testear qué enfoque es el más apropiado a la hora de enseñar la recursividad en asignaturas de primeros cursos de Programación, se han diseñado dos estudios empíricos. En este trabajo describimos en detalle cada uno de ellos, así como los resultados obtenidos y las lecciones aprendidas.

Abstract

One of the main difficulties that students who start learning programming have to overcome is recursion. Our interest is to know the causes behind this problem and try to resolve it. We believe that one of the main reasons that could explain this phenomenon is the enormous conceptual mismatch that they incur when shifting from imperative to declarative paradigm. In order to test which approach is the most appropriate when teaching recursion in subjects of early programming courses we have designed two empirical studies. In this paper we describe in detail each of them, as well as the results and lessons learned.

Palabras clave

Experiencia docente, recursividad, evaluación, diseño cuasi-experimental.

1. Introducción

La adquisición de la competencia de Programación por parte de los estudiantes, principalmente de los primeros cursos de las titulaciones de Informática, es una tarea compleja. Nuestros estudiantes se enfrentan a distintas dificultades a la hora de adquirir destrezas en esta disciplina, esencial para su futuro profesional. Dentro del ámbito de la Programación, conocer y aplicar los principios de la *recursividad* resulta especialmente problemático para ellos.

Las causas que están detrás del fracaso de buena parte de nuestros alumnos a la hora de adquirir habilidades de programación, en general, y de programación recursiva, en particular son de distinto tipo [4,7]. Así se pueden dar problemas derivados de la propia naturaleza de la Programación, que requiere manejar altos niveles de abstracción o usar lenguajes de sintaxis compleja. La falta de motivación o el empleo de métodos de estudio inapropiados por parte de los alumnos (que habitualmente han usado la memorización) también están detrás de dicho fenómeno.

Pero hay dos tipos de dificultades o aspectos en los que nos centraremos y en los que, creemos, podemos influir. Por una parte están las dificultades relacionadas con las *habilidades y actitudes de los estudiantes*, que habitualmente se encuentran desmotivados, no saben bien cómo resolver problemas de programación (y aún menos, de forma abstracta), o no tienen suficiente conocimiento matemático y lógico para abordar los enunciados que se les plantean. Por otra parte, como docentes, hay otro aspecto en el que podemos intervenir, que son los propios *métodos de enseñanza/aprendizaje* empleados en las asignaturas de Programación. Así, por ejemplo, en las clases es habitual usar materiales didácticos de naturaleza *estática*, independientes de quién los

visualiza ni cuándo, mientras que los conceptos que se pretenden transmitir son *dinámicos*, variando en función del tipo del estudiante y, sobre todo, de su momento de aprendizaje. Por otro lado, los docentes suelen dedicar bastante tiempo al lenguaje de programación utilizado y sus detalles sintácticos, en lugar de dedicar gran parte del tiempo y esfuerzo a enseñar a resolver problemas de programación de forma más general y abstracta (con independencia del lenguaje final al que dichas soluciones vayan a ser traducidas). Por lo tanto, como docentes, es innegable la necesidad de conocer las causas que están detrás del fracaso de los alumnos, e introducir cambios y mejoras en nuestros métodos y recursos pedagógicos.

Tan importante como tratar de conocer dichas causas, es validar si, una vez identificadas, los cambios introducidos para solventarlas han supuesto o no en realidad la mejora buscada. Si esta parte de evaluación no se realiza adecuadamente podremos haber generado nuevos problemas o dificultades en el proceso. Normalmente, dicha evaluación se realiza de forma poco exhaustiva o formal porque los docentes habitualmente desconocen técnicas apropiadas para realizar dicho análisis y obtener este *feedback*.

En un trabajo previo, presentado en la anterior edición de JENUI [7], realizamos un estudio empírico que pretendía conocer qué aspectos de la recursividad resultaban más difíciles de asimilar y manejar por parte de nuestros alumnos. Como consecuencia de este trabajo se realizaron algunas modificaciones en los temarios de las asignaturas de programación de primer (*Fundamentos de Programación* I) y segundo curso (*Metodología de la Programación*). Así, en la asignatura de segundo curso se ha añadido un nuevo tema, en el que se repasan y revisan aquellos conceptos que el estudio empírico realizado había identificado como más problemáticos; pero abordándolos, en esta ocasión, desde un punto de vista más abstracto, esto es, desde un enfoque declarativo, haciendo hincapié en el **qué** se hace y aislando todo lo relacionado con el **cómo** se hace. Hay que indicar que, efectivamente, los resultados obtenidos a final de curso en dicha asignatura han sido mejores a los obtenidos el curso anterior (60% de aprobados frente al 25%). Por tanto, incluir este nuevo tema de repaso ha demostrado ser efectivo. Sin embargo, nos centramos ahora en analizar si el hecho de haber repasado dichos conceptos, aplicando estrategias de programación declarativa (frente a las técnicas de enseñanza más clásicas, centradas en un enfoque más funcional) puede estar detrás de dicha mejora. Si esto es así, proponemos utilizar este método de enseñanza de la recursividad desde los primeros cursos y asignaturas de Programación. Creemos, además, que hacerlo puede mejorar la relación de los alumnos con la recursividad, frente al alto rechazo que actualmente esta técnica les produce.

Como aportaciones principales de este artículo están los resultados del estudio empírico realizado (cuyo objeto es testear el "enfoque declarativo vs. imperativo" en la enseñanza de la recursividad), así como el propio método de evaluación empírica empleado, que puede ser tomado como modelo para el estudio del impacto de innovaciones educativas por parte de otros docentes.

2. Evaluación docente mediante un diseño cuasi-experimental

El método tradicional que solemos utilizar los profesores para evaluar el aprendizaje del alumno es el del examen, sistema no exento de inconvenientes, sesgos, errores, etc. Aunque las notas (u otro tipo de resultado académico) sean un criterio válido para verificar el éxito de una innovación didáctica, no hay que olvidar otros efectos que puede merecer la pena medir y evaluar, como son posibles cambios en eficacia, gusto por determinadas asignaturas o temas, percepción del propio aprendizaje, determinadas actitudes o valores, etc [9]. Por eso, la evaluación de los resultados de una innovación docente se debe realizar de una forma lo más científica (y empírica) posible, para que la hipótesis de mejora tenga validez y pueda ser extrapolada a otras situaciones. En este contexto, los *diseños experimentales* permiten establecer causas y justificar explicaciones partiendo de una o varias hipótesis de partida, que se confirman o descartan mediante técnicas de análisis estadístico. En algunos diseños, se parte de la identificación de un grupo de control, generado aleatoriamente, cuyos resultados se toman como referencia del análisis. En los casos en los que no es posible establecer un grupo de control o no hay asignación aleatoria de los sujetos participantes a cada uno de los grupos, se habla de diseño *cuasi-experimental* [9]. Este tipo de diseños son los más adecuados en el caso de las experiencias que se realizan con todos los alumnos presentes en una clase, o los matriculados en un grupo, como es el caso de la que describimos en este trabajo. Para ello, se determina primero la(s) hipótesis de partida del estudio a realizar para poder medir a los sujetos participantes antes y después de un tratamiento en aquella variable o variables en las que se espera que cambien. Así, se distinguen tres fases, *pretest*, *experiencia* y *posttest*, cuyos resultados se analizan posteriormente mediante técnicas de contraste de hipótesis.

2.1 Contexto

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 [1,7,8]. Además, la enseñanza de la recursividad se suele orientar, por regla general, al enfoque concreto del funcionamiento de la recursividad [6,10,11,13,14]. Sin embargo, nuestra hipótesis de partida está en la línea de los autores que inciden más desde el primer momento en una estrategia de pensamiento abstracto [3,5,12,15]. En este sentido, nosotros realizamos una propuesta de *estrategia de pensamiento abstracto* (EPA) [7] en dos etapas: inicialmente, se establecen los elementos que definen el tamaño del problema y se establece el (o los) caso(s) base y su solución(es); en la segunda, se parte de la idea de que el problema está resuelto ya para datos más pequeños, sin pensar en cómo se ha hecho. A partir de ahí, se busca la solución del problema en función de la solución que la recursividad ha proporcionado.

Además, con el fin de afianzar más este esquema declarativo, consideramos que hay que intentar separar la implementación del problema de su definición, entendida ésta como el esquema algorítmico expresado en seudocódigo o en cualquier otro lenguaje (formal, natural, etc.).

2.2. Objetivo e Hipótesis

Nuestro objetivo concreto consiste en confirmar o descartar nuestras creencias mediante el *análisis empírico del uso de cuestionarios y de resolución de problemas en seudocódigo para evaluar el efecto que tiene una Estrategia de Pensamiento Abstracto (EPA) durante el proceso de enseñanza de la recursividad, desde el punto de vista de la motivación del alumno así como de la mejora de su comprensión y de los resultados obtenidos*.

Este objetivo se puede concretar en tres hipótesis de partida:

- *H_0: La resolución de los problemas mediante un esquema algorítmico independiente de la implementación facilita la comprensión de la recursividad*.
- *H_1: Nuestra EPA facilita la comprensión y manejo de la recursividad*.
- *H_2: Nuestra EPA constituye un elemento motivador para el alumno en el aprendizaje de la recursividad*.

A continuación detallamos la experiencia llevada a cabo para confirmar o descartar estas hipótesis.

3. La experiencia realizada

La experiencia la hemos dividido en dos partes, que se corresponden con los distintos momentos temporales en los que se imparte la recursividad en cada una de las asignaturas implicadas: la primera, llevada a cabo en la asignatura de primer curso *Fundamentos de Programación I (FPI)*, que se imparte en el primer cuatrimestre; la segunda, en la asignatura de segundo curso *Metodología de la Programación (MP)*, que se imparte en el segundo cuatrimestre. Aunque los objetivos de las experiencias realizadas en cada una de estas asignaturas han sido distintos, la metodología experimental usada ha sido común, y extrapolable a otras asignaturas y/o innovaciones docentes:

1. En la primera, o fase *pretest*, se recopilan, entre otros, los datos sobre el conocimiento previo de los alumnos participantes, haciendo uso de cuestionarios y de resolución de problemas de forma individual por los alumnos.
2. La segunda, fase *experiencia*, consiste en aplicar la innovación docente que se desea evaluar, en nuestro caso, la impartición de la recursividad de forma declarativa.
3. Por último, en la tercera fase, *posttest*, se vuelven a utilizar cuestionarios y a plantear problemas para que resuelva el alumno.

Puesto que la experiencia es distinta en cada caso, dependiendo de la asignatura, describimos cada una de las dos partes por separado.

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

El objetivo de esta experiencia ha sido el *contraste de la hipótesis H_0*, con la que pretendemos analizar si la separación diseño-implementación en el caso de la recursividad facilita la comprensión de la misma. En la asignatura *FPI*, el tema referido a la recursividad se expone al final de la asignatura, con una duración aproximada de dos semanas (6 horas lectivas). La experiencia se ha realizado en dos grupos de teoría, T1 y T2, considerando el último como grupo de control.

Así, en el **grupo T1**, se intenta separar la definición de la inducción y de la recursividad de su implementación, de forma que el alumno no mezcle los conceptos. La exposición del tema comienza con la explicación del principio de inducción para, a partir de él, definir el concepto de recursividad y usarlo para la definición, por parte del profesor, de los siguientes problemas: la suma de los n primeros números, el factorial de un número n, la potencia de un número, el máximo común divisor de dos números, según el algoritmo de Euclides, el problema de las Torres de Hanoi y la inversa de una palabra. En este punto, se les propone a los alumnos cinco nuevos problemas para que den la definición recursiva de alguno de ellos. En esta parte participan 26 alumnos y los problemas propuestos son:

- *[F1] Sumar dos números enteros m y n*
- *[F2] Sumar los números que hay entre n y m*
- *[F3] Sumar los elementos de una lista*
- *[F4] Decidir si una palabra es un palíndromo*
- *[F5] Obtener el menor elemento de una lista*

El tiempo para esta tarea ha sido de 15 minutos, después de los cuales el profesor ha resuelto el problema *F2*. En la segunda clase, se resuelven los ejercicios *F1*, *F3*, *F4* y *F5* y se les pasa un cuestionario sobre su percepción de la recursividad y de los aspectos que más trabajo les cuesta asimilar. En esta fase participan en 20 alumnos, de los que 1 no había realizado la fase anterior. En la siguiente clase se muestra la implementación en Java de los problemas definidos por el profesor en la primera clase, *sin entrar en el funcionamiento de cada método e incidiendo en la equivalencia entre la definición y la implementación*. Las siguientes dos horas se dedican a la explicación del funcionamiento de la implementación en Java de los problemas propuestos y la última clase se dedica al repaso de los conceptos, se les propone la implementación de los problemas *F2*, *F3* y *F4* y se les pasa otro breve cuestionario sobre las dificultades de la implementación de los problemas recursivos. Para todo ello se les ha dado 20 minutos. El número de alumnos que realizan la experiencia completa es de 9, que son los que consideramos como muestra del grupo T1.

En el **grupo T2**, se sigue la enseñanza tradicional de cursos anteriores, mostrando en paralelo tanto la definición como la implementación de la recursividad. Al finalizar el tema, se pasa un compendio de los cuestionarios 1 y 2 del grupo T1 (al que se le han extraído las cuestiones relativas a la enseñanza de la recursividad mediante el paradigma análisis-diseño-implementación) y que se muestra en la Tabla 5. Además, se les plantea para su resolución los mismos tres problemas (*F2*, *F3* y *F4*) propuestos al grupo T1. En este caso, han sido 14 alumnos los que han participado y han tenido 1 hora para realizar los problemas y el test.

Grupo T1

	Ejercicio F2					Ejercicio F3				
	P	***CCB***	***CB***	***CG***	***T***	***P***	***CCB***	***CB***	***CG***	***T***
Bien	1	1	1	0	**0**	0	0	0	0	**0**
Mal	4	4	4	5	**5**	3	3	3	3	**3**
NC	4	4	4	4	**4**	6	6	6	6	**6**
	Ejercicio F4									
	P	***CCB***	***CB***	***CG***	***T***					
Bien	0	0	0	0	**0**					
Mal	1	1	1	1	**1**					
NC	8	8	8	8	**8**					

Grupo T2

	Ejercicio F2					Ejercicio F3				
	P	***CCB***	***CB***	***CG***	***T***	***P***	***CCB***	***CB***	***CG***	***T***
Bien	14	14	8	8	**4**	6	5	6	5	**4**
Mal	0	0	6	6	**10**	2	3	1	2	**0**
NC	0	0	0	0	**0**	6	6	7	7	**6**
	Ejercicio F4									
	P	***CCB***	***CB***	***CG***	***T***					
Bien	1	1	1	1	**1**					
Mal	2	1	1	1	**1**					
NC	11	12	12	12	**12**					

Tabla 1. Resultados de los ejercicios *F2*, *F3* y *F4* en cada uno de los grupos T1 y T2 de primer curso.

La Tabla 1 contiene los datos relativos a la resolución de los ejercicios resueltos -bien, mal o no contestado (NC)- por ambos grupos. Para cada uno de ellos se distingue si los alumnos definieron bien los parámetros (P), la condición del caso base (CCB), las sentencias correspondientes al caso base (CB), y al caso general (CG) y los alumnos que resuelven completamente bien cada uno de los ejercicios (T).

Los valores de los estadísticos descriptivos (media y desviación típica) de los ítems comunes en los tres cuestionarios realizados en total (dos en el grupo T1 y uno en el T2) son los que se muestran en las dos primeras columnas de la Tabla 5.

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

Como hemos indicado al principio de la sección, la experiencia ha constado de tres fases *(pretest, test y posttest),* que se han llevado a cabo en las semanas 2ª y 3ª del cuatrimestre, durante 4 horas de clase de la asignatura de *Metodología de la Programación*. En ella han participado voluntariamente un total de 65 alumnos de los matriculados en la asignatura: 51 en la primera parte, 52 en la segunda, y 53 en la tercera. Sin embargo, sólo 43 han participado en las tres, habiendo sido éstos los que hemos considerado como muestra. De los 43 participantes, 6 alumnos tienen suspensa alguna de las asignaturas de Programación de primer curso y otros 10 repiten la asignatura. Además, hay que señalar que los alumnos no habían sido informados previamente de lo que iban a hacer en clase, en ninguna de las tres fases, para evitar sesgos.

La fase de ***pretest*** se realizó en los primeros 30 minutos de la 5ª hora de clase de la asignatura y consistió en valorar los ítems del cuestionario (Tabla 2) en una escala de 1 (nada de acuerdo) a 5 (totalmente de acuerdo). También debían indicar su edad, nombre (para la recolección de los datos de las fases posteriores) si repetían o no la asignatura y si tenían o no aprobadas las asignaturas de Programación de primer curso y de Estructuras de Datos, del primer cuatrimestre de segundo curso. Además debían resolver los siguientes ejercicios:

- *[M1] Diseña un algoritmo recursivo que devuelva el menor número de un vector de enteros*.
- *[M2] Diseña un algoritmo recursivo que sume dos números enteros*.

Para resolver el primero de los ejercicios, individualmente, disponían de 15 minutos y para el segundo, de 10; pudiendo elegir el modo que quisieran (esquema algorítmico, lenguaje natural, Java, etc.) para representar la solución.

La fase de **experiencia** se desarrolló en las 2,5 horas siguientes de clase y consistió en plantear los

principios de la recursividad de forma declarativa así como en la resolución de diversos ejercicios sencillos: la suma de los elementos de un vector de enteros, la suma de los elementos de una matriz bidimensional de números enteros, la búsqueda (lineal) de un elemento en un vector y la búsqueda binaria en un vector ordenado. Todos estos problemas habían sido estudiados previamente en su modalidad iterativa.

La fase de ***posttest*** se llevó a cabo en la primera media hora de la siguiente clase y consistió en responder un breve cuestionario sobre su percepción de la EPA (Tabla 3) además de resolver los dos ejercicios siguientes:

- *[M3] Diseña un algoritmo recursivo que invierta una palabra*.
- *[M4] Diseña un algoritmo recursivo que genere una cadena con el contenido de un vector de enteros*.

Los alumnos tuvieron 15 minutos para resolver el primero y, una vez acabado, se les propuso el segundo, para el que dispusieron de 10 minutos. En ambos casos, debían resolverlo individualmente y utilizando el lenguaje que quisieran (seudocódigo, Java, etc.). Las Tablas 2 y 3 muestran los valores estadísticos descriptivos de los ítems de cada uno de los cuestionarios.

N=43	μ	σ
1. Me *gusta* la programación	**3,9**	1,1
2. Al programar, sigo los pasos *análisis - diseño – implementación*	2,9	0,9
3. Me cuesta trabajo *analizar* problemas	2,6	0,9
4. Me cuesta trabajo realizar el *diseño de la* solución	2,9	1,0
5. Me cuesta trabajo la fase de *implementación*	2,4	0,9
6. *Conozco* el principio de *inducción matemática*	2,9	1,2
7. *Comprendo* el principio de *inducción matemática*	2,7	1,2
8. *Conozco* los principios teóricos de la *recursividad*	3,5	0,9
9. *Comprendo* los principios teóricos de la *recursividad*	3,3	1,0
10. Me cuesta trabajo *pensar de forma declarativa*	**2,4**	1,1
11. Me cuesta trabajo *analizar* problemas recursivos	3,0	0,9
12. Me cuesta trabajo *implementar* programas recursivos	3,1	1,0
13. Me cuesta trabajo *implementar* el *caso base*	2,4	0,9
14. Me cuesta trabajo *implementar* el *caso general*	2,9	1,0
15. El conocimiento del funcionamiento de la recursividad me *facilita* la implementación recursiva.	3,2	1,0
16. Necesitaría una *ayuda* para pasar del análisis del problema a la implementación recursiva.	3,1	0,9
17. En general, me *gusta* la recursividad	**2,7**	1,0
18. *Valoro la utilidad* de la recursividad en programación.	**3,8**	1,1

Tabla 2. Ítems del cuestionario *pretest* junto con sus valores estadísticos descriptivos.

N=43	μ	σ
1. Me ha facilitado la c*omprensión* de la *recursividad*	3,3	0,8
2. Me ayuda para *pensar de forma declarativa*	2,9	0,8
3. Me ayuda para *definir* problemas recursivos	3,1	0,7
4. Me facilita la *definición* del *caso base*	3,6	0,8
5. Me facilita la *definición* del *caso general*	3,2	0,6
6. Me facilita la *implementación* de programas recursivos	2,9	0,6
7. Me ayuda para *implementar* el *caso base*	3,3	0,7
8. Me ayuda para *implementar* el *caso general*	3,1	0,6
9. Me *gusta* la recursividad	**2,9**	1,0
10. *Valoro la utilidad* de la recursividad en programación	**3,4**	1,0

Tabla 3. Ítems del cuestionario *posttest* junto con sus valores descriptivos.

La Tabla 4 contiene los datos relativos a la resolución de los ejercicios. Igual que en el caso primer curso, hemos distinguido si han definido bien, mal o han dejado en blanco (NC) la condición del caso base (CCB), las sentencias correspondientes al caso base (CB), y al caso general (CG), los que resuelven bien el ejercicio (T), así como los que resuelven completamente bien los dos (Ambos).

	Ejercicio M1				**Ejercicio M2**				**Ambos**
	CCB	***CB***	***CG***	***T***	***CCB***	***CB***	***CG***	***T***	
Bien	8	4	6	**5**	2	2	3	**1**	**1**
Mal	32	36	34	**31**	27	27	26	**27**	**20**
NC	2	2	2	**2**	13	13	13	**13**	**2**
	Ejercicio M3				**Ejercicio M4**				**Ambos**
	CCB	***CB***	***CG***	***T***	***CCB***	***CB***	***CG***	***T***	
Bien	26	17	3	**16**	19	15	5	**12**	**8**
Mal	15	23	15	**15**	13	16	9	**9**	**12**
NC	1	2	0	**1**	10	11	0	**10**	**0**

Tabla 4. Resultados de los ejercicios *M1*, *M2 (pretest*), *M3* y *M4* (*posttest*) en segundo curso.

En cuanto al lenguaje utilizado para resolver los ejercicios de la fase de *pretest*, el 69% utilizan Java, el 24% seudocódigo y el 7% el lenguaje natural. Sin embargo, en la fase de *posttest*, ninguno utiliza el lenguaje natural, aumentando a 76% los que se decantan por el lenguaje Java y se mantiene el mismo porcentaje de los que utilizan el seudocódigo, siendo el 70% de éstos los que lo utilizan en ambas fases.

4. Análisis de Resultados

4.1. En la asignatura de *Fundamentos de Programación I*

Para evaluar la hipótesis H_o sobre la que se pretendía evaluar si la resolución de los problemas mediante un esquema algorítmico independiente de

su implementación facilita la comprensión de la recursividad, nos fijamos en los valores de la Tabla 5. Los ítems relativos a las dificultades relacionadas con la definición de algoritmos recursivos (ítems 3 al 5 y 10 al 14) parecen mejores en el caso del grupo T1, mientras que los relacionados con la implementación de la recursividad (ítems 6 al 9) parecen mejores en el grupo de control, el T2.

	T1. N=9		*T2. N=14*			
	μ1	**σ1**	**μ2**	**σ2**	**t**	**p**
1. Me *gusta* la programación	3	1,97	**4,21**	0,89	2,02	0,06
2. Al programar, sigo los pasos a*nálisis-definición-implementación*	2,2	1,53	2,79	0,80	1,22	0,24
3. Me cuesta trabajo *analizar* problemas	1,6	1,24	2,57	0,85	2,23	**0,04**
4. Me cuesta trabajo realizar la *definición de la* solución	1,8	1,38	2,62	0,96	2,25	**0,03**
5. Me cuesta trabajo la fase de *implementación*	1,8	1,59	2,79	1,12	1,77	0,09
6. *Conozco* el principio de *inducción matemática*	2,2	1,46	2,86	1,51	1,03	0,31
7. *Comprendo* el principio de *inducción matemática*	2,2	1,5	2,93	1,54	1,12	0,27
8. *Conozco* los principios teóricos de la *recursividad*	2	1,51	3,86	1,17	3,32	**0,003**
9. *Comprendo* los principios teóricos de la *recursividad*	2,1	1,58	3,29	1,20	2,05	**0,05**
10. Me cuesta trabajo *pensar de forma declarativa*	**2,1**	1,57	2,57	1,40	0,75	0,46
11. Me cuesta trabajo *analizar* problemas recursivos	**2**	1,5	3,43	0,94	2,82	**0,01**
12. Me cuesta trabajo *implementar* programas recursivos	2,8	0,8	3,29	1,14	1,12	0,27
13. Me cuesta trabajo *implementar* el *caso base*	2,6	0,96	2,64	0,93	0,09	0,92
14. Me cuesta trabajo *implementar* el *caso general*	2,8	1,12	3,07	0,62	0,75	0,46
15. El conocimiento del funcionamiento de la recursividad me *facilita* la implementación recursiva.	2,9	1,54	3,71	0,99	1,54	0,13
16. En general, me *gusta* la recursividad	2,56	1,28	3,00	1,54	0,89	0,39
17. *Valoro la utilidad* de la recursividad en programación	3,3	0,5	**3,93**	1,00	1,55	**0,05**

Tabla 5. Valores estadísticos descriptivos (***μ*** y **σ)** de los ítems comunes a los cuestionarios de los grupos T1 y T2, junto con los del estadístico **(*t*)** y probabilidad ***(p)*** del contraste de medias de cada ítem.

Sin embargo, si hacemos un contraste de medias de muestras pequeñas e independientes aplicando la *t de Student*, cuyos resultados podemos ver en las columnas (t) y (p) de la Tabla 5, únicamente se considera que existen diferencias estadísticamente significativas (valor de p<0.5) en los ítems 3, 4, 8, 9, 11 y 17. En estos casos, se ha calculado además el valor del tamaño del efecto (***d***) para poder analizar la magnitud del cambio, según las orientaciones más frecuentemente aceptadas de Cohen [2] y su valor se muestra en la Tabla 6.

	μ2- μ1	***d***	***magnitud***
#3	0,97	0,91	grande
#4	0,82	0,69	moderado
#8	1,86	0,60	moderado
#9	1,19	0,84	grande
#11	1,43	1,14	grande
#17	0,6	0,60	moderado

Tabla 6. Valores y magnitud del tamaño del efecto para los ítems cuyas diferencias en medias son estadísticamente significativas.

El análisis de correlaciones de estas variables (Tabla 7) refleja que existe una fuerte correlación (0.73) entre el grupo y el ítem #11 del cuestionario, lo que confirma el hecho de que **a los alumnos del grupo T2 les cuesta mucho más trabajo realizar el análisis de los problemas recursivos.**

	grupo	#3	#4	#8	*#9*	#11
#3	0,51					
#4	0,32	0,57				
#8	**0,62**	**0,65**	0,34			
#9	0,46	**0,64**	0,4	**0,81**		
#11	**0,73**	**0,78**	**0,66**	**0,62**	*0,52*	
#17	0,32	0,07	-0,33	**0,53**	*0,39*	0,16

Tabla 7. Análisis de correlaciones. El valor mínimo significativo para p=0,05 es 0,4133; para p=0,01 es 0,5256 y para p=0,001 es 0,6402.

Como es lógico, a los alumnos que más les cuesta analizar problemas en general (#3), también les cuesta analizar problemas recursivos (#11). De hecho, este ítem (#11) está correlacionado también con el nivel de conocimiento (#8) y comprensión (#9) de la recursividad. También se confirma que los que creen conocer los principios teóricos de la recursividad (#8) valoran la utilidad de la misma en programación (#18).

Por otro lado, si nos fijamos en los resultados de los ejercicios (Tabla 4), los resultados son claramente mejores en el grupo T2. Por tanto, esto permitiría concluir que la resolución de los problemas recursivos mediante la separación definición-implementación en un lenguaje de programación no es más efectiva que la resolución directamente mediante la implementación. Sin embargo, puesto que los del grupo T2 han tenido mucho más tiempo para la resolución de los ejercicios no nos atrevemos a hacer dicha afirmación.

Precisamente esta es una de las limitaciones de la experiencia, junto con el hecho de que en esta experiencia hayan participado distintos profesores para cada grupo. Estos detalles nos han servido para mejorar la experiencia en 2º curso, así como para afianzar el método de evaluación estadística.

Otra limitación ha sido el momento temporal de su realización, a final del cuatrimestre, cuando la asistencia a clase de los alumnos es bastante baja, lo que habrá que tenerlo en cuenta para otros trabajos de investigación similares.

4.2. En la asignatura de *Metodología de la Programación*

La evaluación de la hipótesis H_0 la hemos realizado estudiando la forma en que los alumnos resuelven los ejercicios planteados, ya que tenían libertad absoluta para resolverlo como quisieran: utilizando un esquema algorítmico expresado en seudocódigo o en lenguaje natural, o bien directamente a través de su implementación en Java. Como hemos comentado en la sección 3.2, la gran mayoría de los estudiantes **eligen la resolución directa mediante la implementación en Java**, por lo que la hipótesis H_0 la podemos descartar en 2º curso, en contraposición de lo que ocurría en primero. Esto se explica pues en este curso ya tienen mucha más experiencia en programación y el lenguaje Java es lo suficientemente sencillo como para permitir una solución prácticamente igual a la del seudocódigo.

Para evaluar *objetivamente* la hipótesis H_1 sobre la ayuda que la EPA les supone a los alumnos para comprender y manejar la recursividad, hemos de fijarnos en los valores de la Tabla 4. En ella se observa claramente que los resultados de los ejercicios de la fase de *posttest* son significativamente mejores que los de la fase de *pretest*: de 1 alumno (2%) que en ésta resuelve ambos ejercicios correctamente, en la fase de *posttest* son 8 (19%), y todos los alumnos intentan resolver algo de algún problema. Pero además, los que resuelven el primer ejercicio bien en la segunda fase (16) se triplica con respecto a la fase de *pretest* (5). Más llamativo es el caso del segundo ejercicio, que en la fase de *pretest* únicamente 1 alumno lo resuelve correctamente, frente a los 12 que lo hacen en la fase de *posttest*. Por tanto, podemos intuir que la estrategia ha funcionado.

Además, hemos realizado un análisis de correlaciones[1] entre los datos recogidos (que no incluimos aquí por falta de espacio) para evaluar la hipótesis H_1, en el que se destacan los siguientes resultados:

- Los alumnos que consideran en el *pretest* que el funcionamiento de la recursividad les ayuda a implementar algoritmos recursivos son los que consideran que la EPA les ayuda a definir problemas recursivos (0,32), les facilita la implementación (0,36) y aumentan su gusto por la recursividad (0,31).
- Los que responden correctamente el ejercicio 2.2 son los que han considerado que la EPA les ayuda a definir problemas recursivos (0,34) así como los que mayor gusto por la recursividad han manifestado en el *posttest*.
- Los que consideran que la EPA les ha ayudado a definir el caso base son los que mejor realizan ambos ejercicios de la fase de *posttest* (0,36).

En consecuencia, podemos confirmar la hipótesis H_1 y afirmar que **la estrategia de pensamiento abstracto ayuda a los alumnos a manejar la recursividad.**

La evaluación de la hipótesis H_2 sobre si la EPA constituye un elemento motivador para el alumno en el aprendizaje de la recursividad, nos fijamos en que los valores medios de los ítems 17 y 18 del cuestionario 1 (Tabla 2) y 9 y 10 del cuestionario 2 (Tabla 3) son distintos. Después de realizar un *contraste de medias de muestras emparejadas* (cuyos resultados podemos ver en las columnas (t) y (p) de la Tabla 8), vemos que las diferencias no son estadísticamente significativas (p>0,05). Por tanto, podemos concluir que, per se, **el método no lo perciben motivador**. Cabe preguntarse si dichos valores habrían sido distintos si les hubiésemos enseñado los ejercicios corregidos antes de contestar el test, pues seguramente respondieron bajo el efecto de la incertidumbre de si habían resuelto bien o no los ejercicios. Esta limitación habrá que tenerla en cuenta para experiencias posteriores.

variable		*pretest*	*posttest*	$\mu_{pr}-\mu_{po}$	*t*	*p*
#17, #9	μ	2,72	2,93	**0,21**	1,38	0,17
	σ	1,06	1,01			
#18, #10	μ	3,81	3,44	**-0,37**	2	0,05
	σ	1,15	1,06			

Tabla 8. Valores del estadístico *t* y probabilidad (*p*) obtenidos al hacer el contraste de medias de los ítems #17 y #18 del cuestionario de la Tabla 2 y #9 y #10 de la Tabla 3.

5. Conclusiones y trabajo futuro

En este trabajo se han descrito dos experiencias docentes con alumnos de primeros cursos de Programación, que tenían por objeto analizar empíricamente si enseñar la *recursividad* con un enfoque más declarativo puede favorecer su

[1] El mínimo valor del coeficiente de correlación significativo para una muestra de n=43 es 0,3 para una confianza del 95%; de 0,39 para una del 99% y de 0,49 para una del 999%.

aprendizaje, aplicando esta estrategia de una forma más efectiva e intuitiva.

En la primera experiencia se compararon los resultados de dos grupos de alumnos distintos. Uno de ellos recibió las explicaciones desde un enfoque centrado en mostrar el funcionamiento de la recursividad, mientras que el otro recibió una explicación desde un punto de vista declarativo. Los resultados, contrariamente a lo esperado, arrojaron mejores resultados para el grupo "no declarativo". Sin embargo, hubo varios factores que pudieron afectar a la experiencia y, por tanto, distorsionar los resultados. En primer lugar, la muestra empleada fue pequeña (y distinta en cada grupo); cada explicación fue impartida por un profesor distinto, por lo que el "estilo de enseñanza" de cada uno de ellos podría haber influido en el resultado final; por último, el tiempo dedicado a la resolución de los ejercicios varió de un grupo a otro. Todo ello hace que los resultados obtenidos en este estudio no puedan ser considerados como concluyentes.

Estas limitaciones nos hicieron plantearnos una segunda experiencia, esta vez con un único grupo de alumnos de segundo curso y con la intervención de un único docente. En este caso la muestra fue mayor que en el caso de primero por lo que los resultados obtenidos en esta ocasión tienen mayor validez que los de la primera experiencia, lo que nos permite concluir que explicar la recursividad desde un enfoque más abstracto, aunque no lo perciben como elemento motivador, sí que facilita la comprensión y el manejo de la recursividad. Además, los resultados obtenidos también indican que, mientras que los alumnos de primer curso suelen hacer más uso de esquemas algorítmicos al plantear la solución de los problemas, los alumnos de segundo curso pasan directamente a la implementación en el lenguaje final.

Como trabajo futuro nos planteamos seguir realizando estudios empíricos en asignaturas de Programación, utilizando muestras más amplias de alumnos que aporten mayor validez externa a nuestros resultados, controlando mejor las condiciones en las que se desarrollen. Además, nos gustaría formalizar nuestra metodología de análisis empírico para permitir extrapolar nuestro trabajo a otras asignaturas, y que en este trabajo no hemos podido incluir por falta de espacio.

6. Referencias

[1] Chaffin, A., Doran, K., Hicks, D., y Barnes, T., "Experimental evaluation of teaching recursion in a video game", *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, 2009, vol. 1, no. 212, pp. 79–86.

[2] Cohen, J. (1988). *Statistical power analysis for the behavioral sciences*, Lawrence Erlbaum.

[3] Ginat, D. y Shifroni, E., "Teaching Recursion in a Procedural Environment. How much should we emphasize the Computing Model ?", *ACM SIGCSE Bull*., vol. 31, no. 1, pp. 127–131, 1999.

[4] Gomes, A. y Mendes, A.J., "Learning to program. Difficulties and solutions", *International Conference on Engineering Education – ICEE 2007*, pp. 283-287. (2007)

[5] Gunion, K., Mildford, T., y Stege, U., "Curing Recursion Aversion", *Proc. of the 14th annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, 2009, pp. 124–128.

[6] Hsin, W., "Teaching recursion using recursion graphs", *J. Comput. Sci. Coll*., vol. 23, no. 4, pp. 217–222, 2008.

[7] Lacave, C., Molina, A.I. y Giralt, J., "Identificando algunas causas en el fracaso de la enseñanza de la recursividad. Análisis experimental en la enseñanza de la programación", *Actas de las XIX JENUI*, 2013, pp 225-232.

[8] Mirolo, C., "Learning (through) recursion: a multidimensional analysis of the competences achieved by cs1 students", *Proc. XV Annual Conference on Innovation and Technology in CS Education*, 2010, pp. 160–164.

[9] Morales, P. (2008), *Estadística aplicada a las Ciencias Sociales,* Madrid, Universidad Pontificia Comillas.

[10] Pérez, A., "Sistema Generador de Animacio-nes Interactivas para la Docencia de Algoritmos Recursivos", Tesis Doctoral, URJC, 2011.

[11] Pevac, I., "First experiences with tutor for recursive algorithm time efficiency analysis", *J. Comput. Sci. Coll*., 2012, vol. 28, nº 1, pp. 56-65.

[12] Scholtz, T. y Sanders, I., "Mental Models of Recursion : Investigating Students' Understanding of Recursion", *Proc. XV Annual Conference on Innovation and Technology in Computer Science Education*, 2010, pp. 103–107.

[13] Tessler, J., Beth, B., y Lin, C., "Using Cargo-Bot to Provide Contextualized Learning of Recursion", *Proc. IX Annual International ACM Conference on International Computing Education Research*, 2013, pp. 161–168.

[14] Velázquez, J.A, Pérez, A., y Urquiza, J., "SRec: An Animation System of Recursion for Algorithm Courses", *Proc. XIII Annual Conference on Innovation and Technology in Computer Science Education*, 2008, pp. 225–229.

[15] Wu, C., Dale, N., y Bethel, L., "Conceptual models and cognitive learning styles in teaching recursión", *Proc. XXIX SIGCSE Tech. Symp. Comput. Sci. Educ*., vol. 30, nº1, pp. 292–296, 1998.