Actas de las Jenui, vol. 4. 2019.
Páginas: 55-62

Aplicación del principio KISS a la enseñanza de los fundamentos de programación

Lluís Ribas Xirgo A. Josep Velasco González
Departament de Microelectrònica i Sistemes Electrònics
Universitat Autònoma de Barcelona
08193 Bellaterra
Lluis.Ribas@uab.cat Josep.Velasco@uab.cat

Resumen

Aprender a programar implica no solo conocer un lenguaje en el que describir un algoritmo sino también poder concebirlo. Para facilitar el aprendizaje de la programación y el desarrollo del pensamiento algorítmico en los y las estudiantes que se inician en este campo se puede optar por emplear un subconjunto básico de instrucciones y unos pocos esquemas algorítmicos. En este artículo se presenta el contenido de una asignatura en la que se adoptó esta solución y se analizan los resultados que se han obtenido con ella.

Abstract

Learning to program involves not only acquiring a language in which to describe an algorithm but also being able to conceive it. To make the learning of programming and the development of algorithmic thinking easier for students that are introduced in this field, we use a simple subset of instructions and a few algorithmic schemes. This paper presents the organization of a course that follows this approach, and the results obtained with it are analyzed.

Palabras clave

Algorítmica, didáctica de la programación, enseñanza de la programación a no ingenieros, organización docente, pensamiento algorítmico, programación.

1. Motivación

Programar es un arte que se consigue dominar a través de la experiencia y conociendo muy bien su técnica. Aunque originalmente fuera verdad, la programación se ha convertido menos en un arte y más en una ingeniería de la que hay que conocer muy bien tanto la metodología como la tecnología.

Aprender a programar es complicado porque supone tener que adquirir un pensamiento algorítmico y la capacidad de traducir los algoritmos a programas.

Desgraciadamente, no hay un camino bien definido para llegar a "pensar algorítmicamente" y andarlo supone, además, conocer algún lenguaje para representar dicho pensamiento.

En este sentido, cualquier lenguaje de programación imperativa puede reducirse a unas pocas instrucciones de manera que sea fácil emplearlo para la representación de un algoritmo.

La idea es mantener la simplicidad en todos los procesos que intervienen en la programación para que el esfuerzo de aprendizaje no represente una montaña demasiado alta y escarpada para escalar sino un sendero que alterne algún tramo difícil con otros más fáciles de recorrer.

En este artículo se presenta una experiencia llevada a cabo en la enseñanza de los Fundamentos de la Informática en estudios que no son exclusivamente del campo de la Ingeniería.

Esto no quiere decir que no se pueda aplicar a estudios de grado en Ingeniería, sino que resulta casi un requerimiento de obligado cumplimiento si se pretende que los estudiantes (y *las* estudiantes, aunque en el texto se mantendrá el genérico en masculino) "no informáticos" aprendan a programar.

El grado de Gestión Aeronáutica de la UAB (https://bit.ly/2peMs72) está adscrito a las ramas de Ingeniería y de Ciencias Sociales y Jurídicas, por lo que sus estudiantes pueden tener también un perfil menos técnico. Aun así, tienen que seguir un Plan de Estudios cuyas materias de Ingeniería tienen un peso cercano al 50% del total. De hecho, es un grado oficial heredero de un título propio con el mismo nombre y del ámbito de la Ingeniería que se impartió en la Escuela de Ingeniería de la UAB, tal como se hace actualmente con el grado oficial.

La asignatura de Fundamentos de informática (apartado 2) se concibió como una asignatura común para todos los grados de la Escuela, pero ya desde el primer curso de impartición se vio que la docencia tenía que adaptarse a cada titulación. De hecho, se acabó por disgregarla en tantas asignaturas como

títulos para permitir un mayor ajuste de lo que se enseñaba en cada uno de ellos.

Así pues, el reto era doble: enseñar a programar y, sobre todo, adaptar la enseñanza de la programación a los títulos sin orientación informática.

Se empezó por lo más sencillo, que era enseñar a programar empleando un conjunto muy elemental de instrucciones y esquemas algorítmicos (apartado 4). Aunque esto allanó parte del camino, era necesario que los estudiantes aprendieran a pensar de manera algorítmica (apartado 3).

Con unos exámenes que no han cambiado desde que se empezó a impartir la asignatura, se ha observado como el éxito académico ha aumentado significativamente después de diversos cambios en la asignatura (apartado 6).

Al final, se ha comprobado que la filosofía de la sencillez funciona. De hecho, la mayoría del profesorado en todos los ámbitos, como nosotros, tiende a simplificar el aprendizaje de los alumnos y, por lo tanto, aplica el principio de diseño docente enfocado a la simplicidad denominado KISS, de *Keeping It Simple for Students* (acrónimo usado en [14]).

En otras palabras, que no hay que ser estúpido y emprender una empresa más compleja de la que se puede abarcar, tanto para los profesores como para los estudiantes.

Keep it simple, stupid!

2. Fundamentos de Informática

Se trata de una asignatura de 9 créditos ECTS que supone una media de 225 horas de trabajo para un estudiante, de las cuales 75 son en actividades presenciales.

En la planificación de los grados, se encuentra en el primer semestre del primer año, por lo que es la asignatura en la que se introduce el pensamiento algorítmico y la programación, además de otros conceptos en relación a la arquitectura de los ordenadores y de los sistemas de información, así como de las aplicaciones ofimáticas habituales.

Originalmente, en la asignatura común (curso 2010/11), se impartía en el primer semestre y las prácticas trataban tanto el hardware como el software. Tanto por la elevada carga de trabajo que suponía la asignatura en comparación con las demás como por el hecho de profundizar en temas laterales (hardware) a los del grado, los estudiantes percibieron una asignatura demasiado dura en relación a sus expectativas. De hecho, esto se pudo constatar en el bajo rendimiento académico (38% de aprobados sobre matriculados) que se obtuvo en el primer curso en que se impartió. Sin embargo, cabe decir que el rendimiento académico fue similar a los de otros grupos de la asignatura en las ingenierías.

Aun así, se quiso resolver el problema combinando diversas soluciones.

La primera consistió en rediseñar la planificación de la asignatura para focalizarse en el objetivo básico de la misma: que los estudiantes tuvieran una competencia suficiente en el conocimiento de cómo funcionan los computadores y en la práctica de la programación.

En el curso 2011/12, se eliminó de la parte práctica (en aula y en laboratorio) todo lo relacionado con el hardware y se empezó un proceso de rediseño de las clases de enseñanza de la programación (apartado 4), que se ha ido haciendo de forma progresiva hasta la actualidad.

La segunda, que se implementó en el curso 2012/13, consistió en anualizar la asignatura para que los estudiantes se enfrentaran a los laboratorios prácticos de programación con la experiencia previa del primer semestre. Es decir, se mantuvo el primer semestre con clases de teoría y de problemas y las prácticas se trasladaron al segundo semestre.

Con todo, la estructura de la asignatura de Fundamentos de Informática quedó con una distribución de clases de 3 horas de teoría y 1,5 horas de problemas a la semana durante el primer semestre, y 1,5 horas de prácticas semanales durante el segundo.

Para aprobar, en el primer semestre los estudiantes tienen que entregar sus propuestas de solución al final de cada sesión de problemas y superar dos exámenes parciales. En caso de que no aprueben alguno de ellos, tienen un examen de recuperación al final del semestre.

Con ello se calcula una nota de primer semestre, que es en función de los problemas resueltos que se hayan entregado (evaluación continuada) y de las notas de la primera y segunda parte.

En caso de que sea igual o superior a 5 pueden realizar las prácticas del segundo semestre con opción a aprobar la asignatura. Si no, la nota final de la asignatura será la que hayan obtenido en el primer semestre, aunque hagan las prácticas. La experiencia dice que son pocas personas las que optan por hacer las prácticas sin opción a aprobar la asignatura.

En las clases de teoría se ha optado por un modelo más dinámico y con interacción con los estudiantes que el de la simple exposición: Se han buscado actividades y ejemplos que puedan ser motivadores (apartado 3).

Las clases de problemas se han mantenido en un formato más o menos convencional, presentando uno o dos ejemplos de resolución de problemas y dejando que los estudiantes resuelvan otros parecidos.

En las sesiones de prácticas (apartado 5) se enfrentan, por equipos, al desarrollo de una pequeña aplicación, que van construyendo poco a poco, con la ayuda de una guía y, obviamente, de los profesores de prácticas.

La estructura de los contenidos empieza con una breve introducción a los computadores y la algorítmica para continuar con la introducción a la programación que alcanza su cúspide a medio semestre, con métodos de búsqueda y ordenación. Después se tratan los temas de representación de la información, arquitectura de los computadores, ficheros, bases de datos y aplicaciones ofimáticas.

En este sentido, son contenidos que se repiten en las asignaturas "hermanas" de la Escuela y también en otras universidades.

3. El pensamiento algorítmico

Pensar algorítmicamente es algo poco natural y, por lo tanto, difícil de aprender. En el fondo, se trata de desarrollar una forma de observar los procesos que permita describirlos en un lenguaje muy restringido.

En este sentido, hay propuestas específicas para aprender a sistematizar la obtención de algoritmos a partir de la observación, como la del uso de las cajas negras [2]. Las cajas negras transforman datos de entrada en datos de salida que son conocidos, no así la transformación, que hay que averiguarla. Precisamente esa tarea se acaba convirtiendo en la básica para obtener los algoritmos que transforman unos datos de entrada en unos resultados.

De forma parecida, es posible crear juegos basados no solo en la creación de los algoritmos sino también en ejecutarlos, bien físicamente [4] o bien a través de algún entorno como LOGO [5] o Scratch [12].

Es importante tener presente que la capacidad de pensar algorítmicamente es útil tanto para la programación como para la redacción de manuales de instrucciones, por ejemplo, para actuar en caso de emergencia.

Así pues, es más interesante empezar por practicar con instrucciones simples cuyos intérpretes sean personas en lugar de máquinas.

En este apartado describimos algunos ejercicios que usamos en clase de teoría para acercarnos a este concepto.

3.1. La informática sin computadores

Muchos conceptos de la Informática pueden aprenderse a partir de simples experimentos que, además, aportan un recuerdo tangible y visual, ligado a una experiencia lúdica, que lo hace más duradero. Esta es la clave del aprendizaje.

El proyecto *Informática sin un ordenador* (csunplugged.org/es) hace un compendio de muchas actividades que permiten explicar muchos conceptos de la informática sin emplear ordenadores.

Aunque las actividades están enfocadas a chicos (*y chicas*, recuérdese que empleamos el masculino en las generalizaciones) de edades entre 5 y 14 años, algunas pueden adaptarse para poder realizarse en una clase de teoría a nivel universitario.

Por ejemplo, hay actividades en las que un "robochico" (*kidbot*) tiene que moverse en una cuadrícula (se puede aprovechar el mismo suelo, si es ajedrezado) para alcanzar un objetivo siguiendo un programa con instrucciones muy básicas: girar 90 grados a la derecha o a la izquierda y avanzar a la casilla de enfrente. En el aula, esto se puede reemplazar por una actividad en que, por pares, uno asuma el rol de programador y otro de robot, que dibuja el recorrido en un papel cuadriculado.

Hay que tener en cuenta, además, que muchas de estas actividades describen lo que tienen que hacer los participantes en un lenguaje muy sencillo, con instrucciones claras y simples. Así pues, es otro elemento que puede aprovecharse para la introducción al pensamiento algorítmico en la universidad.

Muchos de los ejercicios que usamos en las clases de teoría vienen inspirados por este proyecto.

3.2. Ejercicios propuestos

Estas actividades se utilizan en las clases de teoría para que los estudiantes tengan una experiencia más vívida (y vivida, sin acento en la primera *i*) de aquello que se cuenta.

No todas las clases incluyen ejercicios de este tipo. En otras se sigue un formato más convencional, aunque procurando que todos los conceptos tengan un ejemplo realista basado en algo cercano a sus estudios.

El objetivo de esta estrategia es que los estudiantes tengan un contexto en el que sea más fácil aprender. De hecho, el uso de distintas técnicas en clase y, en especial, de las que requieren de la participación de los estudiantes, mejoran su atención [1] y crean recuerdos que refuerzan su aprendizaje [3].

En la lista siguiente hay algunos ejemplos de las actividades que se realizan en las clases de Fundamentos de informática.

- *¿Cuántos somos?* Los algoritmos se pueden pensar de muchas maneras y, en este caso, se verá la diferencia entre un método secuencial y uno concurrente. Se les propone que se cuenten y luego se les dice que lo hagan de manera ordenada (si no lo han hecho ya), enumerándose según están sentados. Con ello se les introduce al pensamiento algorítmico. En una vuelta de tuerca, se les presenta un método de conteo en paralelo para que lo ejecuten. Aun siendo más rápido, es habitual que el resultado sea dispar. En cualquier caso, el efecto ilustrativo de la actividad se logra igualmente.

- *Salida del laberinto*. Se trata de una actividad en la que se proporciona a los estudiantes un laberinto con un robot y un destino y otro vacío

(véase la Figura 1). Tienen que escribir en un papel cómo hacer que su robot vaya al objetivo empleando instrucciones elementales y pasarlas a su compañero para que dibuje el recorrido del robot en su cuadrícula vacía. Al final cada uno confronta el recorrido interpretado con el mapa de su compañero para ver si, finalmente, el robot hubiera llegado a su destino.

- *Ordenación por fusión*. Se les pide que se levanten y se vayan al lado del aula que más cerca tienen. Una vez allí, se tienen que ordenar por altura. En cuanto se han formado las dos filas ordenadas, se fusionan para formar una sola lista ordenada. Esta actividad permite, además, hacerles reflexionar sobre cómo se han ordenado para formar las dos filas originales.
- *Personas-bit*. Cada fila de bancos del aula se convierte en un número binario. Los asientos vacíos representan un cero y los ocupados, un uno. Así pues, cada persona tiene que identificar su valor posicional y sumarlo a los valores de sus compañeros de fila para determinar qué número decimal representa su fila en binario. La actividad se puede repetir para números con signo o con una parte fraccionaria (por ejemplo, del pasillo central, si lo hay, hacia un lado).

En general, estas actividades se hacen hacia el final de la clase, cuando la atención decae más. Aun así, hay que tener en cuenta que la atención no es nunca continuada ni tampoco por mucho tiempo [1, 3].

Lo importante es emplear distintos recursos educativos en cada clase para mantener el interés y, sobre todo, que sean efectivas respecto del aprendizaje de los estudiantes.

4. La programación de algoritmos

La transformación de un algoritmo en un programa no queda exenta de dificultad. Hay muchos principios para que este proceso resulte efectivo. El primero de ellos es el de mantener la simplicidad (KISS, de *Keep It Simple, Stupid*[1]), el segundo podría ser el de no hacer nada que no se necesite (YAGNI, de *You Aren't Gonna Need It*) [11] y el tercero, entre otros, el de no repetir código (DRY de *Don't Repeat Yourself*) [6, 15].

No se trata de que los estudiantes tengan que tener estos tres principios en la mente sino de crear un entorno en el que los puedan llevar a la práctica de forma inconsciente. Hay que tener en cuenta que la simplicidad de la programación no es fácil de conseguir cuando hay mucha flexibilidad para codificar un algoritmo [9].

[1] En la literatura se suele cambiar la referencia a *stupid* por otros términos como *straightforward*. Para más información se puede consultar https://en.wikipedia.org/wiki/KISS_principle.

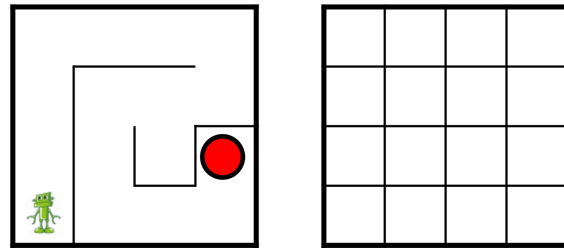

Figura 1: Los laberintos se utilizan en un ejercicio básico de programación.

En nuestro caso, la idea es que el "entorno" incluya el uso de un subconjunto básico de algún lenguaje de programación imperativa.

4.1. La programación 'KISS'

El lenguaje de programación que originalmente se empleó en la asignatura fue C y hace dos cursos se empezó con Python.

El lenguaje C tenía la ventaja de emplear una sintaxis que muchos otros lenguajes comparten, con lo que su aprendizaje facilitaba también el aprendizaje de estos otros lenguajes. Esto era más cierto aun porque solo se empleaba un subconjunto de C. (A modo de ilustración, una de las herramientas que se empleaba en las clases de teoría y problemas para seguir la ejecución de los programas funcionaba con Javascript [8].)

El lenguaje C, sin embargo, adolece de algunos problemas que distraen la atención de los estudiantes en la programación. Por ejemplo, se dedicaba mucho tiempo a los detalles de las funciones `scanf()` y `printf()` para que pudieran emplearse correctamente y, a pesar de todo, seguían siendo una fuente de errores.

Para reducir la dificultad que plantea el lenguaje de programación C en el uso de determinadas funciones y el manejo de las variables se optó por cambiar de lenguaje de programación.

Python, además de su popularidad, es un lenguaje diseñado con unos principios [7] que facilitan la simplicidad de programación. Además, no es necesario aprender tantos detalles en cuanto a las instrucciones de entrada/salida como en el caso del C. Así pues, como en muchas otras asignaturas de introducción a la programación (por ejemplo [13]), se adoptó este lenguaje en nuestra asignatura.

En cualquier caso, la influencia del lenguaje escogido en el aprendizaje de la programación no es muy significativa, como pudimos comprobar al realizar el cambio. Así pues, es mucho más importante la metodología [10].

Ya desde el primer año de impartición de la asignatura se empezó a enseñar a programar manteniendo las cosas lo más simples posible. Cuánto más simples y generales, mejor. De hecho, ya no cabe el recurso a

la eficiencia de ejecución del programa para complicar las cosas: no se trata de aprender a programar con unas restricciones o exigencias añadidas sino, simplemente, de *aprender a programar*.

Así pues, con pocos conceptos basta para empezar: unos "contenedores de valores" que identificamos con un nombre y tres instrucciones para obtener valores del exterior (el teclado), hacer operaciones con ellos y mostrar los resultados al exterior (la ventana de la aplicación en la pantalla).

Se sigue con las instrucciones de ejecución alternativa (`if`) e iterativa (`while`) y ya está. Para ayudarles en el uso de las iteraciones, se les proporcionan dos esquemas algorítmicos: el del recorrido y el de la búsqueda. Todos los programas los construyen a partir de secuencias de estos esquemas algorítmicos.

El procesado de listas (y cadenas de caracteres y vectores y matrices, incluso) así como de ficheros de flujo se explica también a partir de combinaciones de esquemas algorítmicos, alternativas y asignaciones.

Se utilizan las funciones para organizar el código en bloques más pequeños que eviten el anidamiento de los esquemas. De hecho, solo se hace una excepción en los algoritmos de ordenación y de operaciones con matrices y por motivos más estéticos que prácticos, ya que podrían programarse como recorridos convencionales con un único bucle.

Este estilo de programación permite que los estudiantes puedan concentrarse en los aspectos básicos de la algorítmica y de la programación.

Reducir los algoritmos a secuencias de instrucciones de entrada/salida, cálculos y alternativas y de esquemas de recorrido y búsqueda permite "pensar algorítmicamente".

La persona tiene que determinar qué datos necesita el algoritmo para poder obtener el resultado y, si se trata de datos estructurados, si el resultado depende de todos ellos (recorridos) o de uno solo (búsqueda).

Evidentemente, se combina esto con un proceso de diseño descendente, según el problema a resolver. Los primeros problemas no requieren de ninguna profundidad en el diseño y, después de introducir las funciones, se proponen problemas con una profundidad de diseño de uno o dos niveles, a lo sumo. Hay que tener en cuenta que tienen que ser problemas que se resuelvan en media hora, incluyendo el algoritmo y su programación.

5. Las prácticas de programación

Las prácticas se realizan en el segundo semestre y consisten en el desarrollo de una pequeña aplicación que tiene que ver con los Estudios.

En general, es durante las prácticas donde más aprenden a programar, puesto que, de una parte, se insiste más en el funcionamiento del programa que en la parte algorítmica y, de la otra, hay un seguimiento más individualizado por parte del profesorado.

Los estudiantes reciben un guion explicativo de las tareas que deben de acometer en cada sesión.

Antes de cada sesión, un miembro de cada equipo tiene que presentar, en un minuto, lo que ha hecho el equipo desde la sesión anterior y lo que hará durante la sesión. La nota de estas "presentaciones a la puerta" o *door pitches* es individual y los dos miembros de cada equipo tienen que repartirse las presentaciones de manera equitativa, no necesariamente de forma alternativa.

Para conseguir la máxima atención posible, durante la sesión no se pueden utilizar los móviles y, obviamente, los ordenadores sólo pueden emplearse para cuestiones relacionadas con la práctica, incluidas las consultas en Internet. Hay una nota de actitud en el laboratorio, también de carácter individual, que disminuye si se realizan actividades no vinculadas a la práctica en el laboratorio.

Finalmente, cada equipo tiene que presentar el estado de su programa al profesor al acabar la sesión. Esta nota es de equipo.

Este tipo de evaluación se realiza durante las diez primeras sesiones y se dejan las dos últimas para las defensas de los proyectos.

Las notas de las primeras sesiones son binarias para facilitar el proceso de evaluación.

Estas defensas consisten en una presentación, por parte del equipo, de la aplicación que han desarrollado, seguida de una breve entrevista a cada uno de los miembros del equipo.

La nota final de las prácticas se calcula ponderando la nota de la defensa y la nota de la evaluación continua. Los pesos de cada parte pueden variar según el curso, dependiendo de cuántas sesiones evaluadas se hayan hecho. En cualquier caso, y para que la nota refleje las competencias de cada persona, quién no supere la entrevista individual no puede aprobar las prácticas. Hay, eso sí, una sesión adicional de reevaluación para aprobarlas, con una nota menor.

Desde que se implantó este sistema, no obstante, solo un pequeño porcentaje de los estudiantes que realizan las prácticas tiene que asistir a la prueba de reevaluación.

De hecho, hay muy pocas personas que suspendan las prácticas, cosa que ilustra tanto la importancia de reducir la relación entre número de profesores y alumnos como la validez del principio de simplicidad para mejorar el aprendizaje de la programación.

Respecto de lo primero, hay que tener en cuenta que las prácticas de laboratorio se hacen en el segundo semestre y que aquellos alumnos que han suspendido el primero sin posibilidad de aprobar la asignatura ya no las hacen. Antes de la introducción del cambio en el curso 2012/2013, un 95% de los estudiantes matriculados en la asignatura realizaban las prácticas,

mientras que la media de participación en ese año y hasta la actualidad es del 68%. Esto reduce el tamaño de los grupos de laboratorio de 24 personas por grupo a 16 y, en consecuencia, aumenta el tiempo de atención de los equipos por parte del profesorado.

En cuanto a la simplicidad en la programación, hay que tener en cuenta que no se introducen más elementos de programación que los ya vistos en el primer semestre.

6. Resultados académicos

Una de las cosas que se ha mantenido constante en la asignatura de Fundamentos de informática es el formato de los exámenes escritos, por lo que es posible averiguar qué efecto han producido los cambios que se han ido introduciendo en la docencia de la asignatura desde el curso 2010/11.

Los exámenes consisten en una serie de ejercicios de programación y uno o dos de representación de la información (sistema binario) en el que corresponde a la parte en que se ha tratado este aspecto. Los demás contenidos de la asignatura se cubren con preguntas de respuesta abierta. Estas preguntas han ido cambiando con los años para adaptarse a la distinta intensidad con que se trataban estas partes de la materia. En cualquier caso, la complejidad de los exámenes no ha variado desde el primer curso en que se empezó a impartir la asignatura, ni tampoco ha habido cambios significativos en la manera de evaluarlos. Hay que tener en cuenta que los profesores responsables de la asignatura han sido siempre los autores de este artículo.

En el curso 2010/11, la estructura de la asignatura era común para todos los grados de la Escuela. Durante el mismo curso ya se vio la imposibilidad de seguir la misma planificación que en los grupos de otros títulos, por lo que se simplificó el tratamiento de aquellos temas no tan esenciales para el grado de Gestión Aeronáutica. Es decir, todo lo referente a la Arquitectura de los Computadores y a los Sistemas Operativos se redujo a lo básico para entender el funcionamiento de los ordenadores.

En el curso siguiente (2011/12), aun con esta adaptación, el rendimiento académico fue algo peor incluso que en el 2010/11. En parte, esto se explica porque los repetidores en el curso 2010/11 venían del grado de ingeniería propio en Gestión Aeronáutica y, por lo tanto, con un perfil más tecnológico, mientras que en el curso 2011/12, los alumnos de nueva entrada y parte de los repetidores provenían tanto de bachilleratos tecnológicos como de Ciencias Sociales.

Se optó por hacer dos cambios más: una introducción a la programación desde el pensamiento algorítmico, manteniendo la filosofía de la simplicidad, y posponer al segundo semestre las prácticas de la asignatura.

El efecto de estos cambios en los resultados se puede apreciar en el rendimiento académico obtenido en el curso 2012/13. A partir de ese curso, los resultados han ido oscilando entre, aproximadamente, el 50% y el 70%, tal como se puede apreciar en la Figura 2.

Nos hubiera gustado ver un rendimiento académico ascendiente fruto de algunos cambios en la forma de contar las cosas, pero lo cierto es que no hay un vínculo claro entre esos cambios y el rendimiento académico obtenido.

Por ejemplo, en el curso 2012/13, se introdujo Scratch como herramienta tanto para el pensamiento algorítmico como de práctica en la programación simplificada y se descartó al curso siguiente a raíz de

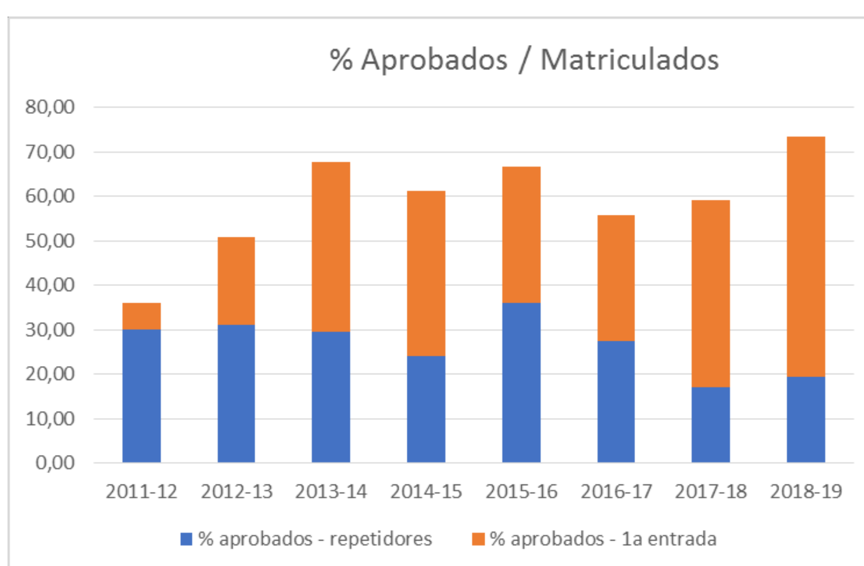

Figura 2: Evolución de los resultados académicos desde 2011/12.

una encuesta en la que los estudiantes, de forma anónima, respondieron qué elementos les ayudaron más en el aprendizaje de la programación. Les hubiera gustado aprovechar ese tiempo con C, que era el lenguaje de programación que se empleaba en las prácticas. En el curso siguiente, se eliminaron las actividades en Scratch.

Se ha experimentado también con el uso de vídeos en clase, tanto para ilustrar conceptos de teoría, como para mantener la atención de los estudiantes. Sin embargo, es otro elemento que se abandonó a raíz de una encuesta en la que indicaban que no les ayudaba demasiado a recordar los conceptos de la teoría.

En el curso 2017/18 se cambió de C a Python para eliminar la complejidad del primero en cuanto a las instrucciones de entrada/salida y la dificultad que presenta para ocultar determinados detalles de programación. Sin embargo, tampoco se ha observado un incremento en el rendimiento académico sino más bien lo contrario.

Creemos que esta disminución puede ser debida a que los estudiantes que repetían la asignatura, que contaban con experiencia en C, tuvieron menos éxito académico en ese curso.

El uso de Python, sin embargo, se ha mantenido porque es mucho más fácil de emplear en cualquier plataforma o de ejecutarse en web, sin instalaciones de ningún tipo.

El promedio del éxito académico desde el 2012/13 supera el 60%. Esto se ha traducido en una reducción del número de alumnos constante desde el curso 2013/14, pasando del máximo de 133 en el curso 2011/12 hasta los 83 del curso 2018/19, con una entrada media de 66 nuevos alumnos por año.

7. Conclusiones

La enseñanza de los Fundamentos de la Programación es un reto más complejo cuando los aprendices no tienen un perfil técnico y los estudios no son exclusivamente de Ingeniería. Este es el caso de la asignatura de Fundamentos de informática del grado de Gestión Aeronáutica, inscrito tanto en el ámbito de las Ingenierías como de las Ciencias Sociales y Jurídicas.

Para facilitar la adquisición del pensamiento algorítmico y el aprendizaje de las técnicas de programación, se diseñó la docencia de manera que las clases de teoría fueran un compendio de presentaciones más o menos magistrales, demostraciones prácticas y actividades que requirieran la participación del alumnado. Las clases de problemas se mantienen con un formato más convencional, en el que las y los estudiantes tienen la oportunidad de trabajar en la programación de algoritmos muy similares a los casos planteados en las clases de teoría.

El hecho de haber simplificado el lenguaje algorítmico y la programación reduce el *síndrome de la página en blanco* y, sobre todo, permite que el o la estudiante se concentre en el método y no en los detalles.

Sin embargo, tal como se puede observar en los resultados, el mayor efecto en la mejora del éxito académico se consiguió anualizando la asignatura: el hecho de reducir la carga de trabajo del alumnado en el primer semestre hizo también posible que, en los casos en que hubiera dificultades, tuvieran un poco más de tiempo para superarlas.

Al final, la enseñanza de la programación tiene que acompañar a las y los estudiantes en su aprendizaje de la mejor manera posible. Según nuestra experiencia, esto pasa por darles tiempo para experimentar y por proporcionarles unas herramientas simples en las que crear los algoritmos y programarlos.

Referencias

[1] D. M. Bunce, E.A. Flens y K.Y. Neiles. How long can students pay attention in class? A study of student attention decline using clickers. *Journal of Chemical Education*, 87 (12), 1438–1443, 2010.

[2] Martin Cápay y Martin Magdin. Alternative methods of teaching algorithms. En *2nd World Conf. on Educational Technology Researches. Procedia – Social and Behavioral Sciences*, 83, pp. 431 – 436, 2013.

[3] Benedict Carey. *How we learn. The surprising truth about when, where, and why it happens*. Penguin Random House, junio 2015.

[4] Gerald Futschek y Julia Moschitz. Developing algorithmic thinking by inventing and playing algorithms. En *Constructionism 2010, The 12th Eurologo Conference*, Paris, agosto 2010.

[5] Juraj Hromkovic, Tobias Kohn, Dennis Komm y Giovanni Serafini. Examples of algorithmic thinking in programming education. En *Olympiads in Informatics*, 10, pp. 111 – 124, Vilnius, 2016.

[6] Andrew Hunt y David Thomas. *The Pragmatic Programmer*, Addison-Wesley, 2000.

[7] Tim Peters. *The Zen of Python*. Agosto, 2004. Disponible en https://www.python.org/dev/peps/pep-0020/

[8] Lluís Ribas Xirgo. Animación interactiva de algoritmos para cursos de introducción a la programación. En *Actas de las XVIII Jornadas de Enseñanza Universitaria de Informática, JENUI*, pp. 503 – 510, Sevilla, julio 2011.

[9] Sander Rossel. KISS—One best practice to rule them all. Why KISS isn't easy. En *Simple programmer*, agosto 2015. Disponible en:

https://simpleprogrammer.com/kiss-one-best-practice-to-rule-them-all/

[10] Rosana Satorre, Patricia Compañ y Faraón Lloveras. Un modo de entender la programación. En *Actas de las X Jornadas de Enseñanza Universitaria de Informática, JENUI*, pp. 387 – 391, Alicante, julio 2004.

[11] Stephen R. Schach. *Object-Oriented and Classical Software Engineering*. Séptima edición, McGraw-Hill, 2007.

[12] Ricardo de J. Botero Tabares. Revisión bibliográfica de los juegos digitales para el aprendizaje de la programación orientada a objetos. *Cuaderno ACTIVA*. No. 4, pp. 91–106, Medellín, julio-diciembre 2012.

[13] José A. Troyano, Fermín Cruz, Mariano González, Carlos G. Vallejo y Miguel Toro. Introducción a la programación con Python, computación interactiva y aprendizaje significativo. En *Actas de las XXIV Jornadas de Enseñanza Universitaria de Informática, JENUI*, pp. 223 – 230, Barcelona, julio 2018.

[14] Charles F. Urbanowicz. “Mnemonics, quotations, cartoons, and a notebook: ‘tricks’ for appreciating cultural diversity”. Capítulo *de Strategies In Teaching Anthropology*, editado por Patricia Rice and David McCurdy, pp. 132-140, New Jersey, Prentice-Hall, 2000.

[15] Greg Wilson, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumbley, Ben Waugh, Ethan P. White y Paul Wilsonet. Best Practices for Scientific Computing. En *PLOS Biology* 12 (1): e1001745, 7 de enero de 2014.