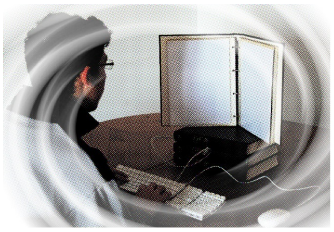


ReVisión

Artículo invitado

AENUI

Asociación de Enseñantes Universitarios de la Informática

«El año que viene estudio día a día»: intentando que los alumnos cumplan la promesa*

Marco Antonio Gómez Martín, Pedro Pablo Gómez Martín

Departamento de Ingeniaría del Software e Inteligencia Artificial

Universidad Complutense de Madrid

Madrid

marcoa@fdi.ucm.es, pedrop@fdi.ucm.es

Resumen

La evaluación contínua es un intento para que los alumnos distribuyan a lo largo de todo el curso el estudio de las asignaturas, en vez de condensarlo en los periodos de examenes. En muchas ocasiones esa evaluación contínua se ha llevado a cabo mediante pruebas evaluadas intermedias que no son más que pequeñas réplicas de los examenes que se encontrarán al final de la asignatura. En este artículo presentamos un modelo alternativo para la asignatura de *Estructuras de Datos y Algoritmos* en el que la evaluación continua se realiza utilizando directamente los ejercicios propuestos en los apuntes de clase. Para evitar el sobreesfuerzo por parte del profesor, utilizamos como apoyo el software que se utiliza en los concursos de programación.

Palabras clave: Evaluación continua, corrección automática, concursos de programación.

1. Introducción

La implantación del EEES ha supuesto la entrada en las universidades de la evaluación continua, que exige un seguimiento cuidadoso de la evolución del aprendizaje de los alumnos. Uno de los objetivos de la evaluación continua es conseguir que el alumno distribuya el tiempo que invierte en la asignatura a lo largo de todo el curso académico, en vez de condensar la mayor parte del esfuerzo al final, cuando se acercan los periodos de exámenes.

En general, la evaluación continua se ha implantado incorporando pequeñas pruebas evaluadas a lo largo del curso que suponen un porcentaje concreto de la nota final o con entregas de prácticas o trabajos suplementarios. La así llamada "nota de la evaluación continua" es entonces ponderada junto con la nota del examen final de acuerdo a unos porcentajes determinados de antemano en la ficha docente. Como es evidente, estas tareas evaluadas adicionales suponen un esfuerzo extra por parte del profesorado, tanto por la necesidad de preparar las tareas a las que enfrentar a los alumnos, como por su corrección. Para intentar minimizar el esfuerzo adicional necesario, los docentes nos hemos puesto a buscar mecanismos que consigan reducir el tiempo total necesario [11]. Herramientas de corrección automática, test de unidad o autoevaluación son distintos ejemplos que se están utilizando hoy en día.

Desde el punto de vista de un alumno "responsable" que sin evaluación continua era capaz de planificar su trabajo y hacer esa distribución de esfuerzo dedicado a la asignatura correctamente, la evaluación continua supone un trabajo adicional: no es suficiente con ir estudiando la materia que se va impartiendo y haciendo los ejercicios propuestos, sino que, además, hay que realizar esas otras tareas evaluables.

En este artículo describimos nuestro intento de unificar las actividades que en un modelo sin evaluación continua querríamos que los alumnos distribuyeran durante el curso (estudio y realización de ejercicios propuestos) con las realizadas en la evaluación continua.

En concreto, los alumnos de la asignatura Estructura de Datos y Algoritmos son enfrentados todas las semanas a una evaluación utilizando uno de los ejercicios propuestos en el tema que se ha estado impartiendo durante la semana anterior. De esta forma, si el alumno ha dedicado tiempo a la asignatura conseguirá superar el ejercicio fácilmente.

Para que la carga de trabajo del profesor no se incremente demasiado, en la experiencia hemos utilizado software de gestión de concursos de programación para automatizar la evaluación de esos ejercicios.

*Financiado por el Ministerio de Educación y Ciencia (TIN2009-13692-C03-03)

2. Concursos de programación

Existen a nivel mundial numerosos concursos de programación dirigidos a estudiantes de informática, tanto en el ámbito universitario como en el ámbito de la educación secundaria. Aunque a nivel nacional este tipo de concursos no es demasiado conocido, en otros países las universidades llegan incluso a ofertar asignaturas optativas cuyo objetivo es preparar a los alumnos para enfrentarse a los problemas típicos que aparecen en ellos. El concurso más conocido internacionalmente es el ACM-ICPC (*International Collegiate Programming Contest*) cuyos orígenes se remontan a 1970 [7]. En él, la resolución de los problemas implica conocimientos avanzados de estructuras de datos, algoritmos sobre grafos o geometría computacional entre otros.

La relevancia a nivel mundial de este concurso está fuera de toda duda. En el año 2009, los concursos asociados al ICPC (final mundial y regionales) involucraron a 7109 equipos de 1838 universidades distintas. Y estos datos son sólo la punta del iceberg. Cada concurso regional engloba a un territorio que puede llegar a incluir una gran cantidad de Universidades. Dado que el aforo en los concursos regionales está limitado, muchas de ellas se ven forzadas a realizar clasificaciones locales para decidir qué equipos las representarán acudiendo a los concursos regionales oficiales. Skiena estima que al menos 85 000 estudiantes participaron en 2009 en esas competiciones locales ([14], prefacio). A estos datos hay que sumarles los del resto de concursos de programación no directamente relacionados con ICPC, como las olimpiadas internacionales de informática (IOI) o los concursos organizados por entidades privadas como Google o Tuenti.

El tipo de problemas en los concursos tipo ICPC es siempre similar: se basan en realizar programas que lean de la entrada estándar datos que se procesan para escribir por la salida estándar la respuesta. Así por ejemplo, el programa puede leer la descripción de un grafo (vértices y sus aristas) y escribir el número de componentes fuertemente conexas que contiene. El enunciado suele incluir un ejemplo de entrada y un ejemplo de salida para complementar la descripción textual. En el momento en el que el equipo envía su solución, esta es compilada y ejecutada contra un conjunto de casos de prueba (grafos, en el ejemplo) que se mantienen en secreto y que validan la solución aportada. Dado que el objetivo principal es poner a prueba el conocimiento de algoritmia y estructuras de datos, la ejecución se puede restringir tanto en tiempo como en memoria, para forzar a realizar implementaciones de algoritmos con complejidades específicas. No se valora, sin embargo, la calidad del código; el uso de variables globales, la ausencia de sangrado o de comentarios en el código, o el uso intensivo de macros son pasados por alto.

Aunque originalmente los problemas eran entregados a través de dispositivos de almacenamiento extraíbles, hoy en día se hace uso de *sistemas de gestión de concursos*, que son aplicaciones software que facilitan el control. PC^2 (*Programming Contest Control*[1]) ha sido la aplicación de referencia durante años; hoy hay otras alternativas, basadas en Web, como DOMJudge[2] o Mooshak[3], ambos con licencias abiertas.

Los gestores de concursos permiten a los jueces humanos definir los problemas del concurso, así como los casos de prueba que utilizará cada uno y configurar sus límites de ejecución. Durante el concurso, los participantes envían sus soluciones a través de la red, y reciben los veredictos automáticamente. Los jueces humanos pueden en todo momento controlar el estado del concurso, ver el código fuente de los envíos, contestar aclaraciones y, en algunos casos, modificar, si hay algún problema, los veredictos automáticos.

La disponibilidad de este tipo de aplicaciones permite a cualquier institución organizar un concurso de programación sin tener que preocuparse de la infraestructura software. Sin embargo, es necesario poblarlos de contenido, añadiéndoles los problemas y casos de prueba correspondientes.

Una alternativa a instalar un juez propio es hacer uso de las páginas y portales de distintas instituciones que proporcionan colecciones de problemas y evaluadores automáticos previo registro. El más conocido mundialmente es español, mantenido en la Universidad de Valladolid[4] y que a día de hoy supera los 3000 problemas en su colección, tiene más de 12 millones de envíos y más de 500 000 usuarios con algún problema solucionado.

3. Evaluación continua

Con las nuevas titulaciones de grados, parcial o totalmente implantadas en nuestras universidades, se ha adoptado un enfoque de evaluación que, al estar orientada al proceso, debe ser continua, lo que permite a los estudiantes (y profesores) tener información permanente sobre su progreso.

Esta evaluación continua tiene como consecuencia directa un aumento en la carga de trabajo de los profesores, que tienen que dedicar tiempo y esfuerzo no sólo a la preparación de las clases, sino también a esas actividades de evaluación continua y a su calificación.

Con los años, han sido muchas las alternativas que distintas experiencias docentes han puesto a prueba. En el entorno de las asignaturas de programación, la corrección automática de prácticas está muy extendida [2, 9]. Otras estrategias pasan por utilizar wikis [1], autoevaluación [6] o concursos [3, 5] o incluso tecnologías que permiten editar, depurar y ejecutar el código sobre Moodle [8].

La experiencia que describimos en este artículo está ligada con los concursos de programación descritos anteriormente. Es en realidad argüible si introducir a los alumnos en un entorno competitivo es bueno o no; de hecho todo parece indi-

[1] http://www.ecs.csus.edu/pc2/
[2] http://domjudge.sourceforge.net/
[3] http://mooshak.dcc.fc.up.pt/
[4] http://uva.onlinejudge.org/

car que depende del método de aprendizaje particular de cada alumno y lo preparado que se esté para la competición [4, 13].

Para evitar la polémica y la posible queja por parte de los alumnos más desfavorecidos en un entorno competitivo, nuestra aproximación no es la de realizar concursos durante la clase y utilizar sus resultados como método de evaluación, sino utilizar únicamente los tipos de ejercicios y las herramientas disponibles para su organización.

En concreto, y como veremos con detalle en la sección siguiente, lo que hemos hecho ha sido *traducir* ejercicios que típicamente se encuentran en las hojas de problemas asociados a un tema teórico de programación al estilo de los concursos de programación (con su entrada y salida bien especificada). Utilizando DOMJudge como herramienta de gestión de los casos de prueba y de los envíos, enfrentamos a los alumnos a esos problemas y evaluamos su rendimiento no en el contexto de competición contra otros alumnos, sino en el de elaboración y entrega de ejercicios.

4. Experiencia en Estructuras de Datos y Algoritmos

En segundo curso del plan de estudios de los tres grados relacionados con informática de nuestra universidad aparece como asignatura *Estructura de Datos y Algoritmos* (EDA). Se trata de una asignatura anual de 9 créditos. El primer cuatrimestre versa fundamentalmente sobre análisis de la eficiencia de los algoritmos, diseño y análisis de algoritmos iterativos y recursivos y algoritmos de ordenación.

Durante el segundo cuatrimestre (donde se sitúa nuestra experiencia) se abordan los tipos abstractos de datos (TAD), implementación y uso de tipos de datos lineales, arborescentes, tablas asociativas y dos últimos temas sobre divide y vencerás y vuelta atrás.

Para todo lo anterior utilizamos C++ como soporte de forma que, sin perder de vista la componente más formal, se implementan los distintos algoritmos y TAD para que los alumnos puedan trasladar lo visto en clase a un ejecutable funcionando.

Para la evaluación, se realiza un examen parcial en febrero, con el que los alumnos pueden liberar los contenidos del primer cuatrimestre, y un examen final en junio. Además, un tercio de la calificación final proviene de las notas sacadas de las actividades de evaluación continua realizadas durante el año.

Desde el punto de vista de infraestructura, los 9 créditos suponen tres horas de clase a la semana. Dos de ellas se imparten de forma colectiva mediante las tradicionales clases magistrales; la tercera hora semanal queda para clases de problemas, ya sea en aula o en un laboratorio.

De cara a la evaluación continua del segundo cuatrimestre existen alternativas variopintas que han sido puestas en marcha por los profesores de los distintos grupos y que van desde la resolución en casa y sobre papel de ejercicios, hasta la elaboración y entrega de pequeñas prácticas en grupos como la generación y análisis de gráficas de tiempos de algoritmos de ordenación. Otros ejemplos son la resolución individual de problemas en la pizarra o pequeños exámenes periódicos realizados en el laboratorio.

En nuestro caso, y siguiendo con la máxima de que es preferible que la evaluación continua esté formada por pequeñas actividades de no más de una hora y entregas frecuentes [12], hemos recogido datos *todas* las semanas aprovechando esa tercera hora de laboratorio. A final de curso habíamos realizado así doce actividades de evaluación continua independientes.

Para poder evaluar esas doce actividades a más de 60 alumnos sin que el tiempo dedicado a ello se dispare hemos utilizado las técnicas de concursos mencionadas antes. El funcionamiento general es el siguiente.

- Al comienzo de cada tema de la asignatura se dejan disponibles los apuntes del tema desarrollados por el equipo de profesores, así como una hoja de ejercicios relacionados.

- Durante las clases teóricas se va haciendo referencia a esos ejercicios e incluso resolviendo (en C++) algunos de ellos, y se empuja a los alumnos a que intenten los demás. Aquellos que intentan hacer los ejercicios sobre el papel y tienen dudas pueden preguntar sobre ellos por anticipado (antes de verlos en clase) en tutorías.

- Cada semana alguno de esos ejercicios mencionado en clase es *traducido* al formato de ejercicios típico de concurso. Los alumnos no conocen de antemano cuál de ellos será, pero tienen la certeza de que el problema al que se enfrentarán en el laboratorio aparece en la hoja.

- Durante la hora de laboratorio, los alumnos intentan solucionar, en grupos de dos, el ejercicio en C++. Para ello pueden/deben utilizar los TAD implementados en clase. Para entregar la solución utilizan DOMJudge que evalúa inmediatamente la validez de la solución.

- El profesor, durante esa hora, está comprobando los envíos para hacer una inspección manual del código y dar el visto bueno final a la misma. Además, a pesar de ser una actividad evaluada, responde dudas y soluciona problemas que los alumnos puedan tener, pues consideramos que es en el momento en el que el alumno se enfrenta a un problema cuando más posibilidades hay de que asimile los conceptos necesarios para su solución.

- Los primeros minutos de la clase de teoría siguiente es utilizada para comentar la solución correcta al ejercicio. La aplicación web utilizada para los envíos se deja abierta de forma que, aunque fuera ya de la evaluación continua, los alumnos puedan enviar soluciones alternativas o intentar enviar una solución correcta si no lo lograron a tiempo.

Dado que al final del cuatrimestre se termina con doce notas (hay alguna semana más de clase, pero se pierden algunos días de laboratorio por fiestas y otras causas) y la evaluación continua constituye un tercio de la nota final, cada uno de los ejercicios repercute en algo menos de tres décimas en la nota final del cuatrimestre o menos de décima y media en la nota final del curso. Es por esto que no nos planteamos la evaluación *fina* de cada ejercicio: el problema está bien (pasa los casos de prueba y el código cumple unos mínimos, utilizando los TAD vistos en clase, etc.) o está mal.

Con la naturaleza de la prueba en donde la solución del alumno debe funcionar correctamente para todos los casos de prueba, lo que le estamos exigiendo al alumno son dos cosas:

- que sepa describir correctamente el algoritmo y estructuras de datos necesarias,
- que lo sepa expresar en el lenguaje de programación concreto (en este caso C++), es decir, que el programa compile, se ejecute y sea capaz de pasar los casos de prueba expresados en un formato concreto.

El hecho de que la evaluación de cada uno de los ejercicios sea completamente binaria puede conducir a la conclusión de que podemos estar dando demasiada importancia a la segunda parte: quizá un alumno no sea capaz de conseguir pasar la prueba por algún detalle menor de implementación, incluso cuando el algoritmo que ha ideado es el adecuado. Esta sensación parece reforzarse si se supone que en nuestras pruebas de evaluación continua mantenemos el control de los concursos de programación, donde los participantes se enfrentan a los problemas sin ayuda. La realidad, sin embargo, es distinta; durante la hora que dura cada prueba, los profesores están sirviendo de apoyo, respondiendo dudas y solucionando los problemas "menores" de programación que los alumnos puedan sufrir, tales como dificultades con el entorno, errores de sintáxis, dificultad en la interpretación de la entrada del programa o en la generación de la salida. Esta monitorización del trabajo de los alumnos se utiliza también dentro del proceso de evaluación: si la solución que está desarrollando el estudiante no es la adecuada (por ejemplo está utilizando un algoritmo iterativo en vez del recursivo que se ha estudiado durante la semana) el estudiante es informado; si insiste y envía el algoritmo incorrecto, la solución es anulada e ignorada en el juez. La revisión *in situ* también tiene otra ventaja con respecto a la detección de copias. Dado que se va revisando el código en el momento de la entrega, es fácil detectar envíos sospechosamente parecidos con otros que el profesor recuerda haber visto unos momentos antes.

Es justo mencionar que la ayuda suministrada a los alumnos respecto a los detalles de programación no está exenta de polémica entre el profesorado. A los alumnos, aparte de los conocimientos nuevos de la asignatura, lo único que se les está pidiendo es que sean capaces de programar en C++ y seguir un formato estricto en la entrada y la salida de sus programas, algo que aprenden en primero. Hay quien considera que proporcionarles ayuda en estos aspectos es restar importancia a la necesidad de que tengan habilidad y agilidad programando.

Polémicas aparte, y volviendo al software de gestión de concursos y la evaluación automática, su uso hace que el esfuerzo que el profesor tiene que realizar para cada una de estas actividades ocurra siempre *antes y durante* la propia actividad, y nunca *después*[5]. Es en la fase de preparación cuando se debe traducir el ejercicio a un formato apto para el juez, creando la solución de referencia y generando unos casos de prueba lo suficientemente exhaustivos como para estar seguros de que una solución es correcta si funciona con todos ellos. Dependiendo del problema concreto, esto puede llevar más o menos tiempo, pero en nuestra experiencia nunca supuso más de dos horas (y en general necesitábamos menos de una).

Veamos de forma detallada uno de los ejercicios utilizados para comprender mejor el proceso. El problema se sitúa dentro del tema de TAD arborescentes, que se centra en la implementación y uso de árboles binarios implementados utilizando una estructura jerárquica de nodos y manejando punteros. En la parte de teoría se explican los conceptos relacionados y se desarrolla el código C++ que gestiona esa estructura así como recorridos de pre/in/postorden y por niveles; el tema termina con los conceptos e implementación de árboles de búsqueda. Dentro de este contexto, uno de los ejercicios que aparecen al final del tema dice lo siguiente:

> Implementa una operación en los árboles de búsqueda que *balancee* el árbol. Se permite el uso de estructuras de datos auxiliares.

El enunciado utilizado en la evaluación continua (Figura 1) parte de una lista de elementos ordenados (lo que serían las claves del árbol de búsqueda original en el ejercicio de la hoja de problemas) y debe construir un árbol balanceado. La forma de comprobar que esa construcción es correcta consiste en pedirles que escriban su recorrido en preorden y postorden, lo que determina de manera unívoca la estructura del árbol construido (para los casos en los que los elementos no están repetidos).

En frío, el ejercicio parece demasiado complejo para afrontar su programación completa en tan sólo una hora. Sin embargo, los alumnos tienen a su disposición implementado el TAD lista y el TAD árbol (con sus recorridos). La solución únicamente debe por tanto preocuparse de hacer un uso correcto de esas estructuras, lo que requiere menos de 100 líneas de código.

Plantear problemas cuya solución sea corta no es sólo importante para que los alumnos puedan realizarlos en el tiempo asignado, sino para que la carga por parte del profesor no sea excesiva. Como se ha dicho, antes de realizar la actividad es necesario implementar la solución de referencia que, en este caso, no llevó más de 10 minutos. Algo más laboriosa es la creación de los casos de prueba que deben discernir entre las

[5]Más allá de pasar a la hoja de cálculo correspondiente qué grupos han realizado correctamente el ejercicio

soluciones correctas e incorrectas. Para poder crear casos de prueba exhaustivos, normalmente se programan generadores que los construyen. En este caso, la aplicación contenía 80 líneas de código, y escribía listas desde 1 hasta 50 elementos, luego algunas listas con números repetidos, y terminaba con dos listas de números grandes de más de 50 000 elementos.

Como referencia, a continuación se describen brevemente los ejercicios realizados en todas las sesiones de ejercicios durante el cuatrimestre, y su objetivo dentro del curso.

1. *Dígitos*: dado un número escribir sus dígitos separados y la suma de los mismos. Por ejemplo para el 3433, escribir `3 + 4 + 3 + 3 = 13`. No se permitía usar recursión, de forma que obligatoriamente tenían que hacer uso del TAD `Pila` visto en clase.

2. *Paréntesis balanceados*: dada una cadena con paréntesis, corchetes y llaves, decir si estos están perfectamente balanceados o no. También exige el uso de pilas.

3. *¿Quién empieza?* (problema de Josephus): el problema tiene una ambientación en la que estamos al frente de un grupo de niños colocados en círculo. Debemos elegir a uno de ellos, y usamos el procedimiento de ir retirando del círculo uno de cada n hasta dejar sólo a uno, que será el elegido (y la salida del programa). La solución pone a prueba el uso de colas.

4. *Números afortunados*: una variación del anterior también utilizando colas. Se quita uno de cada dos, luego uno de cada tres, uno de cada cuatro, etc., hasta que sólo queda uno.

5. *James Bond*: se describe un algoritmo de cifrado y descifrado de mensajes y se pide, dado un mensaje cifrado, descrifrarlo. El método exige el uso de pilas y colas de caracteres.

6. *Árboles de Fibonacci*: se definen los árboles de Fibonacci y se pide, dado un n pintar (utilizando un formato parecido al utilizado por comandos tipo `tree`) el árbol asociado a ese n. Pone a prueba los conceptos de árboles binarios y su construcción recursiva.

7. *De-reconstrucción*: dados los recorridos en preorden e inorden de un árbol de enteros no repetidos, escribir su recorrido en postorden. Requiere la construcción del árbol a partir de los dos recorridos (manejando listas y recursión).

8. *Ductilidad de árboles binarios*: igual que el anterior, pero en vez de darles preorden e inorden, se les daba inorden y postorden.

9. *Me pilló el toro*: ejercicio para utilizar árboles de búsqueda, almacenando en él las notas agregadas de la evaluación continua de distintos alumnos para generar la lista final por orden alfabético.

10. *Balanceo*: explicado anteriormente (véase Figura 1).

11. *Elévame*: implementar la exponenciación rápida utilizando divide y vencerás. Para evitar el desbordamiento, debía calcularse *módulo* 31543 (que es un número primo).

12. *Números de Fibonacci*: parecido al anterior pero con exponenciación de matrices 2×2 para el cálculo rápido de números de Fibonacci.

5. Resultados

Desde el punto de vista del trabajo del profesor creemos que este método de evaluación continua es *cómodo*, en el sentido de que el cuatrimestre termina con doce actividades evaluadas sin suponer una carga de trabajo excesiva. De hecho, la carga de trabajo es la misma independientemente del número de alumnos por lo que desde un punto de vista estadístico, es más eficiente cuantos más alumnos haya matriculados. Además, tampoco creemos que, aún con el número tan grande de actividades, estemos cayendo en una sobre-evaluación [10], pues lo que estamos evaluando son ejercicios que los alumnos conocen de antemano y que por tanto deberían haber formado parte de su trabajo personal.

Sin embargo, los resultados para los alumnos no han sido todo lo positivos que nos gustaría. La experiencia nos ha presentado una realidad abrumadora y preocupante: nuestros alumnos de los grados en informática *no están acostumbrados a programar*. Todos los alumnos eran de segundo curso por lo que se les supone al menos un curso de *Fundamentos de la Programación*. Somos conscientes de que algunos de ellos podían no tenerla aprobada, pero al menos deberían tener la soltura suficiente como para poder enfrentarse a las pequeñas aplicaciones que les pedíamos (en nuestras titulaciones en primero también se utiliza C++). Muchos de ellos tenían problemas para hacer compilar su código o incluso en el uso del entorno de desarrollo. Alguno incluso llegaba a rebelarse porque, al no ser capaces de poner correctamente condiciones de bucles, alegaban que eso era temario «del año pasado» y no de este. Ni siquiera la primera actividad evaluada que consistía en escribir los dígitos de un número fue resuelta por todos los alumnos.

Otro aspecto que también nos llamó la atención fue el poco esmero y dedicación en hacer aplicaciones que funcionen *en todos los casos*. Dada la masificación de los grupos de primero, en la asignatura de programación se dedica poco tiempo en comprobaciones exhaustivas de casos límite o gestión de errores. Eso hace que lleguen a segundo con la filosofía de hacer aplicaciones que funcionen para los casos “normales” (en nuestro caso aquellos que aparecen como ejemplo en el enunciado) sin preocuparse de los casos especiales o conflictivos (aquellos ocultos en los casos de prueba del juez). Era normal tener que defender la validez de los casos de prueba para hacerles entender que el problema no estaba en ellos sino en su solución.

Los comentarios de los alumnos llegaron a extremos sorprendentes. Los ejercicios 7 (*De-reconstrucción*) y 8 (*Ductilidad de árboles binarios*) son muy similares, y en lugar de ver el segundo como una oportunidad de conseguir una buena puntuación, muchos lo vieron como una decisión intencionada para perjudicarles.

Creemos que la experiencia ha sido positiva, pese a esa apreciación negativa por parte de los alumnos. Estimamos que para evitarla se debe hacer más hincapié en la necesidad de que *programen* las cosas que se ven en clase. Y es recomendable que, para que puedan hacerlo, tengan accesibles las colecciones de ejercicios utilizadas en años anteriores.

La Figura 2 muestra una vista parcial de la tabla con los envíos de los alumnos a lo largo de todas las evaluaciones. Cada fila representa a uno de los equipos y cada columna a un problema. Los problemas evaluables están numerados del 1 al 12 con la salvedad de que el problema número 2 es en realidad el llamado `Parentes`. A continuación aparece una columna `Train??` por cada problema, lo que les permitía hacer envíos tardíos de los ejercicios para probar sus soluciones. El orden de los equipos no debe interpretarse de ninguna forma (están ordenados por número de problemas resueltos *en total*, incluyendo los de entrenamiento, que quedan fuera de la imagen).

Las celdas en verde marcan que el equipo consiguió enviar una solución correcta (los números interiores tienen sentido en el contexto de un concurso pero no son usados en nuestro caso). Las celdas rojas indican equipos que hicieron algún intento de resolver el problema pero no lo consiguieron. Como puede verse, incluso en la "zona alta de la tabla" hay muchas celdas en blanco, lo que implica que el grupo no fue capaz de tener una solución que pasara los casos de ejemplo del enunciado y por tanto no intentaron ningún envío.

6. Evaluación de la evaluación contínua

Aunque de manera imprevista, la experiencia ha servido para poner de manifiesto que desde primero debería acercarse a los alumnos mucho más hacia los ordenadores. En general la mayor parte de nuestros alumnos tiene poca o ninguna simpatía a estar delante del ordenador poniendo a prueba los conceptos adquiridos. Creemos que es algo que deberían hacer por ellos mismos sin necesidad de empujarles, pero dado que no parecen tener esa tendencia, tenemos que encontrar los mecanismos para que lo hagan. Una posible vía es el uso de los jueces en línea como el de la Universidad de Valladolid comentado anteriormente. A pesar de que muchos de los problemas que hay en él tienen una dificultad que escapa a lo que un alumno de primero puede resolver, es cierto que hay muchos (tal vez unos pocos cientos) que sí están a su alcance. Además, contienen enunciados motivadores o entretenidos, que los hacen mucho más apetecibles que los habituales en las hojas de ejercicios de primero.

La evaluación continua consiste, como decíamos al principio del artículo, en un cambio de paradigma en el que pasamos a evaluar *el proceso*. La evaluación continua pura, por tanto, utilizaría directamente las pruebas incrementales semanales para calificar a los alumnos. Nosotros hemos utilizado sin embargo una aproximación mixta en la que la calificación conseguida con la evaluación continua es un *complemento* a la nota final alcanzada mediante un examen tradicional.

Curiosamente, esto pone a prueba a la propia evaluación continua, que debe estar de alguna manera relacionada con el resto de mecanismos de evaluación y ser coherente con ellos. En concreto, debería medir aspectos similares de la disciplina a los que mide el examen final, aunque sea mediante herramientas diferentes. Dicho de otro modo, la evaluación continua debería permitir predecir la nota final de los alumnos; en otro caso, no estaríamos diseñando correctamente los mecanismos de evaluación. A modo de ejemplo, si en nuestra asignatura, a pesar de haberles pedido programar en C++ durante las pruebas de evaluación continua, en el examen les pidiéramos, por ejemplo, verificación formal de los algoritmos implementados, la relación de ambas calificaciones sería, presumiblemente, muy pequeña. Los ejercicios fueron, por tanto, similares a los realizados en la evaluación continua y en muchos casos podrían formado parte de ella. A modo de muestra, uno de los ejercicios del examen era:

> Diseñar recursivamente, y sin usar estructuras de datos auxiliares ni el iterador de la clase, un método de la clase `Arbus`[6] que dada una clave *k* que se sabe está en el árbol, devuelva la siguiente clave en orden creciente. Calcular el coste de dicho método.

El número de alumnos no es lo suficientemente grande como para que los resultados sean estadísticamente significativos, por lo que no hemos realizado un análisis en profundidad de la relación entre las notas de la evaluación continua y la del examen final. En cualquier caso, sí es posible identificar un gran grupo de alumnos que mantienen un resultado similar en ambas calificaciones, aprobando (o suspendiendo) las dos con notas similares.

También hay algunos casos en los que se observa un ascenso en la nota conseguida en el examen final frente a la de los ejercicios semanales. Suponemos que esto es debido a que, sencillamente, hay alumnos que estudian al final e ignoran la evaluación continua. Normalmente son los alumnos más capacitados los que se pueden permitir el lujo de dejar para el final prepararse las asignaturas y, aun así, aprobar. Curiosamente son ellos los grandes perjudicados por el proceso de adaptación al EEES, dado que la manera de evaluar les exige un *trabajo contínuo* que ellos no hacen por preferir (y ser capaces de) concentrar todo el esfuerzo al final del cuatrimestre y aun así superar las asignaturas.

[6] Es la clase vista durante el curso que implementa en C++ los árboles de búsqueda.

Queda un tercer grupo de alumnos que, habiendo conseguido calificaciones positivas durante la evaluación continua, suspenden en el examen final. Son estos alumnos los que ponen en duda si los ejercicios semanales y el examen final sirven realmente para comprobar las mismas capacidades o si, por el contrario, los profesores hemos planteado mal alguna de las dos cosas.

Analizando nuestros resultados, hemos encontrado a tres alumnos en esa situación, dos de ellos con diferencias de calificación extremadamente altas. Un estudio un poco más cuidadoso desvela la causa. Como se ha comentado, los ejercicios semanales son resueltos *por grupos* de dos alumnos. En los tres casos, el *compañero* había sacado una calificación en el examen acorde a la conseguida en la evaluación continua.

7. Conclusiones y trabajo futuro

El artículo ha descrito el uso de software de gestión de concursos de programación, disponible de forma gratuíta, para la evaluación continua de los alumnos en la asignatura de *Estructuras de Datos y Algoritmos,* en segundo curso del Grado de Informática. Obviando la puesta en marcha del sistema (instalación y registro de los usuarios), el esfuerzo por parte del profesor es manejable y escala perfectamente independientemente del número de alumnos matriculados. Es necesaria una dedicación previa a cada tarea planteada a los estudiantes que en ninguno de los 12 ejercicios planteados durante la experiencia ha supuesto más de dos horas.

Los resultados han sido satisfactorios en lo referente a la utilidad de la evaluación y a la relación con el resultado de los alumnos en el examen final. Aunque de manera informal, se ha comprobado que en general aquellos alumnos que demostraban haber dedicado tiempo a los ejercicios semanales se veían compensados con una calificación positiva en el examen final. En los casos que no ha sido así, los datos han demostrado que se ha debido a que la calificación positiva conseguida durante la evaluación continua era debida al compañero de grupo.

A pesar de estos resultados, la recepción de este mecanismo de evaluación por parte de los alumnos ha sido desigual. Las mayores quejas han puesto de manifiesto una falta de agilidad programando que deberían haber adquirido en primero. Esto nos lleva a plantearnos la idea de generalizar el uso de sistemas de gestión de concursos de programación con ejercicios pensados específicamente para los alumnos de los primeros cursos. En esta línea estamos creando un sistema similar al juez en línea de la Universidad de Valladolid, pero que contendrá una colección de ejercicios en español pensados para la docencia. En él, dejaremos disponibles los enunciados de los ejercicios desarrollados durante esta experiencia y durante la organización de las diferentes ediciones del concurso de programación para ciclos formativos ProgramaMe[7]. Esperamos con él animar a los alumnos, desde primero, a *aprender a programar programando*, de manera autónoma gracias a la corrección automática de sus soluciones.

[7] `http://www.programa-me.com`

Referencias

[1] M. Arevalillo-Herráez, R. Pérez-Muñoz y Y. Ezbakhe: *Evaluación automática de aportaciones en un sistema basado en wikis.* En Actas de las XVI Jornadas de Enseñanza Universitaria de la Informática, (JENUI 2010), pp. 59–66, Santiago de Compostela, julio de 2010.

[2] M. A. Gómez Martín, G. Jiménez Díaz y P. P. Gómez Martín: *Test de unidad para la corrección de prácticas de programación, ¿una estrategia win-win?* En Actas de las XVI Jornadas de Enseñanza Universitaria de la Informática, (JENUI 2010), pp. 51–58, Santiago de Compostela, julio de 2010.

[3] G. Jimenez-Diaz, J. A. Recio-Garcia, B. Diaz-Agudo y G. Flórez-Puga: *Uso de competiciones y sistemas de clasificación como metodología de evaluación de una asignatura.* En Actas de las XVIII Jornadas de Enseñanza Universitaria de la Informática, (JENUI 2012), pp. 25–32, Ciudad Real, julio de 2012.

[4] R. Lawrence. *A new learning paradigm: competition supported by technology*, capítulo Motivating students using competitive programming, pp. 11–40. Sello editorial, 2010.

[5] A. Martínez-Usó y P. Garcia-Sevilla: *Concurso de proyectos en la asignatura "Percepción Visual" del máster en sistemas inteligentes.* En Actas de las XVIII Jornadas de Enseñanza Universitaria de la Informática, (JENUI 2012), pp. 105–112, Ciudad Real, julio de 2012.

[6] E. Mosqueira-Rey. *La evaluación continua y la autoevaluación en el marco de la enseñanza de la programación orientada a objetos.* En Actas de las XVI Jornadas de Enseñanza Universitaria de la Informática, (JENUI 2010), pp. 223–230, Santiago de Compostela, julio de 2010.

[7] W. B. Poucher y M. A. Revilla. *From Baylor to Baylor*. Lulu Enterprises, 2009.

[8] J. C. Rodríguez del Pino, E. Rubio y Z. J. Hernández. *VPL: Laboratorio Virtual de Programación para Moodle.* En Actas de las XVI Jornadas de Enseñanza Universitaria de la Informática, (JENUI 2010), pp. 429–435, Santiago de Compostela, julio de 2010.

[9] F. P. Romero, J. Serrano-Guerrero y H. Pérez de Inestrosa. *CUESTOR: Una nueva aproximación integral a la evaluación automática de prácticas de programación.* En Actas de las XVI Jornadas de Enseñanza Universitaria de la Informática, (JENUI 2010), pp. 493–500, Santiago de Compostela, julio de 2010.

[10] J. Serrano-Guerrero, F. P. Romero y J. A. Olivas. *La sobre-evaluación: efectos negativos de una mala planificación de la evaluación.* ReVisión, vol. 4, núm. 2, julio de 2011.

[11] F. Sánchez Carracedo, J. J. Escribano Otero, M. J. García García, J. González Rodríguez y E. Millán Valldeperas. *Ideas para reducir el trabajo del prof-EEES-or.* En Actas de las XVI Jornadas de Enseñanza Universitaria de la Informática, (JENUI), pp. 301–308, Santiago de Compostela, julio de 2010.

[12] M. Valero y J. J. Navarro. *Una colección de metáforas para explicar (y entender) el EEES.* En Actas de las XVI Jornadas de Enseñanza Universitaria de la Informática, (JENUI 2010), pp. 293–300, Santiago de Compostela, julio de 2010.

[13] E. Verdú y R. M. Lorenzo. *A new learning paradigm: competition supported by technology*, chapter Effective Strategies for Learning: Collaboration and Competition, pp. 11–40. Sello editorial, 2010.

[14] E. Verdú, R. M. Lorenzo, M. A. Revilla y L. M. Regueras. *A new learning paradigm: competition supported by technology*. Sello editorial, 2010.

Marco Antonio Gómez Martín es profesor del Departamento de Ingeniería del Software e Inteligencia Artificial de la Universidad Complutense de Madrid. Desde siempre ha estado interesado en la calidad de la docencia impartida, como demuestra que su primer artículo como postgraduado fuera en unas JENUI. Tras aquel primer trabajo vinieron muchos otros en distintas conferencias nacionales e internacionales relacionadas con la docencia de la informática. Su tesis doctoral giró en torno a videojuegos educativos y en la actualidad sus intereses siguen en la línea de videojuegos y del uso de sus mecánicas para la enseñanza.

Pedro Pablo Gómez Martín fue profesor de la Facultad de Informática de la Universidad Complutense entre 2001 y 2005, y lo ha vuelto a ser desde 2011, tras unos años dando clase en Formación Profesional. Además, es profesor del Máster de Videojuegos de la UCM desde su creación en 2004. Sus intereses de investigación se centran en la enseñanza de la programación (por ejemplo a través de concursos, como ProgramaMe) y en la arquitectura e inteligencia artificial en videojuegos.

Balanceo

Es sabido que para ciertos algoritmos sobre árboles binarios es preferible que éstos estén balanceados. Se entiende por un árbol balanceado aquél en el que la talla de sus dos hijos no difiere en más de una unidad y, además, los dos hijos están a su vez balanceados.

Tan importante es la propiedad de estar balanceado que ciertos árboles garantizan que se mantienen balanceados ante inserciones reestructurándose si es necesario. En el momento en el que se detecta que la inserción de un nuevo elemento ha desbalanceado el árbol, modifica su posición y la de sus nodos cercanos para que, manteniendo el mismo recorrido en inorden, el árbol siga estando balanceado.

Una alternativa más costosa es realizar ese balanceo *a posteriori* en vez de hacerlo de forma incremental en el momento de las inserciones. Esta operación, que requiere memoria adicional, consiste en obtener el recorrido en inorden del árbol y construir desde cero un árbol nuevo de forma que el resultado final esté balanceado (y mantenga el mismo recorrido en inorden).

Entrada

Cada caso de prueba consiste en una única línea de números enteros terminados con un `-1`. La lista de números representa el recorrido en inorden de un árbol que hay que balancear. Ten en cuenta que el recorrido *no* incluye el `-1`.

La entrada terminará con un recorrido en inorden vacío, que no generará salida.

Salida

Para cada caso de prueba se escribirán dos líneas, una con el recorrido en preorden y otra con el recorrido en postorden del árbol balanceado.

Si existen varias opciones para obtener el árbol balanceado, se elegirá aquella en la que el hijo izquierdo tenga un nodo más que el hijo derecho.

Pon una línea en blanco para separar cada caso de prueba.

Entrada de ejemplo

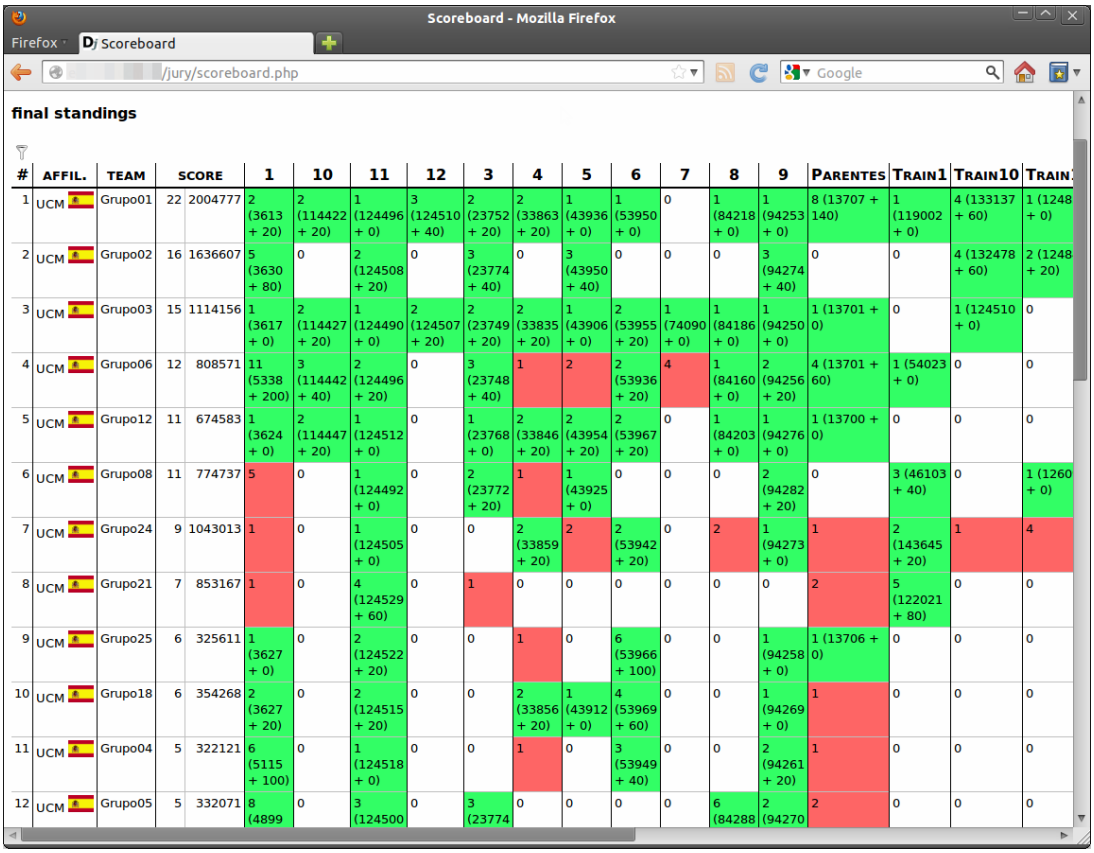
```
1 2 3 4 -1
1 2 3 4 5 -1
-1
```

Salida de ejemplo

```
3 2 1 4
1 2 4 3

3 2 1 5 4
1 2 4 5 3
```

Figura 1: Enunciado utilizado en la evaluación continua adaptado de un problema de la hoja de ejercicios.



#	AFFIL.	TEAM	SCORE	1	10	11	12	3	4	5	6	7	8	9	Parentes	Train1	Train10	Train
1	UCM	Grupo01	22 2004777	2 (3613 + 20)	2 (114422 + 20)	1 (124496 + 0)	3 (124510 + 40)	2 (23752 + 20)	2 (33863 + 20)	1 (43936 + 0)	1 (53950 + 0)	0	1 (84218 + 0)	1 (94253 + 0)	8 (13707 + 140)	1 (119002 + 0)	4 (133137 + 60)	1 (1248 + 0)
2	UCM	Grupo02	16 1636607	5 (3630 + 80)	0	2 (124508 + 20)	0	3 (23774 + 40)	0	3 (43950 + 40)	0	0	0	3 (94274 + 40)	0	0	4 (132478 + 60)	2 (1248 + 20)
3	UCM	Grupo03	15 1114156	1 (3617 + 0)	2 (114427 + 20)	1 (124490 + 0)	2 (124507 + 20)	2 (23749 + 20)	2 (33835 + 20)	1 (43906 + 0)	2 (53955 + 20)	1 (74090 + 0)	1 (84186 + 0)	1 (94250 + 0)	1 (13701 + 0)	0	1 (124510 + 0)	0
4	UCM	Grupo06	12 808571	11 (5338 + 200)	3 (114442 + 40)	2 (124496 + 20)	0	3 (23748 + 40)	1	2	2 (53936 + 20)	4	1 (84160 + 0)	2 (94256 + 20)	4 (13701 + 60)	1 (54023 + 0)	0	0
5	UCM	Grupo12	11 674583	1 (3624 + 0)	2 (114447 + 20)	1 (124512 + 0)	0	1 (23768 + 0)	2 (33846 + 20)	2 (43954 + 20)	2 (53967 + 20)	0	1 (84203 + 0)	1 (94276 + 0)	1 (13700 + 0)	0	0	0
6	UCM	Grupo08	11 774737	5	0	1 (124492 + 0)	0	2 (23772 + 20)	1	1 (43925 + 0)	0	0	0	2 (94282 + 20)	0	3 (46103 + 40)	0	1 (1260 + 0)
7	UCM	Grupo24	9 1043013	1	0	1 (124505 + 0)	0	0	2 (33859 + 20)	2	2 (53942 + 20)	0	2	1 (94273 + 0)	1	2 (143645 + 20)	1	4
8	UCM	Grupo21	7 853167	1	0	4 (124529 + 60)	0	1	0	0	0	0	0	0	2	5 (122021 + 80)	0	0
9	UCM	Grupo25	6 325611	1 (3627 + 0)	0	2 (124522 + 20)	0	0	1	0	6 (53966 + 100)	0	0	1 (94258 + 0)	1 (13706 + 0)	0	0	0
10	UCM	Grupo18	6 354268	2 (3627 + 20)	0	2 (124515 + 20)	0	0	2 (33856 + 20)	1 (43912 + 0)	4 (53969 + 60)	0	0	1 (94269 + 0)	1	0	0	0
11	UCM	Grupo04	5 322121	6 (5115 + 100)	0	1 (124518 + 0)	0	0	1	0	3 (53949 + 40)	0	0	2 (94261 + 20)	1	0	0	0
12	UCM	Grupo05	5 332071	8 (4899	0	3 (124500	0	3 (23774	0	0	0	0	6 (84288	2 (94270	2	0	0	0

Figura 2: Tabla (parcial) con los envíos realizados por los alumnos.