Actas de las Jenui, vol. 3. 2018.
Páginas: 247-254

Ordocoordinación: Cómo Organizar 700 Estudiantes en un Nuevo Campus (y no Morir en el Intento) *

Jorge Castro, Xavier Farreres, Joaquim Gabarro,
Pilar Nivela, Antoni Perez-Poch*, Elvira Pino, Jose Miguel Rivero
Departament de Ciències de la Computació
* También: Institut de Ciències de l'Educació
Universitat Politècnica de Catalunya, Barcelona Tech, 08034 Barcelona
{castro,farreres,gabarro,nivela,aperez,pino,rivero}@cs.upc.edu

Resumen

A partir del Q1 de 2017 el Departamento de CS de la UPC se ocupa de la docencia de Informática I en el nuevo campus de la EEBE. Dicha docencia ha sido singular en dos aspectos. Primero, hubo que organizarla al mismo tiempo que se impartía. Segundo, todos los números son grandes. En el Q1 de 2017, hubo 686 estudiantes matriculados, 18 docentes y se prepararon 108 exámenes de laboratorio. Para tratar con estas singularidades hemos adoptado una forma de coordinación a la que hemos llamado *ordocoordinación*. Es una coordinación flexible y rápida en la que los docentes generan y consensúan un conjunto mínimo de reglas. Es de abajo a arriba y requiere una toma de decisiones ágil, por lo que el número de emails ha sido importante. En Q1 de 2017: $350 \times 18 = 6300$ emails. Creemos que esta aproximación merece ser explicada y que puede ser aplicada a otras asignaturas.

Abstract

Since Autumn Term 2017 the Department of Computer Science of the Universitat Politecnica de Catalunya UPC-BarcelonaTech is in charge of teaching "Fundamentals of Programming" in the new Diagonal-Besós Campus, at EEBE School. This new endeavour had to face two particular challenges: First, due to organizing constraints, it had to be organized at the same time it was being first taught. Second, all the numbers involved were large. In effect, in Autumn Term 2017, 686 students enroled, with a teaching staff of 18 instructors, and 108 laboratory tests being prepared. To deal with these challenges, we agreed to coordinate ourselves in a particular way which we name *ordocoordination*. We define ordocoordation as a flexible and quick particular way of coordination in which teachers generate and agree on a minimum set of rules. It is a bottom-up procedure, requiring taking quick decisions. As a consequence of applying this particular coordination, the number of sent emails has been a large one: in Autumn Term 2017: $350 \times 18 = 6300$ emails were interchanged. We believe that this approach deserves to be reported, and also that it is relevant to other subjects.

Palabras clave

Informática I, ordocoordinación, coordinación por reglas, muchos grupos, validación automática, Python.

1. Introducción

En Septiembre de 2016, la UPC tras recuperarse levemente de la Gran Recesión, inauguró un nuevo Campus de Ingeniería en Barcelona[1]. Dicho Campus se halla situado a caballo entre Barcelona y Sant Adrià del Besòs. Es una zona del área metropolitana de Barcelona de fuertes contrastes, uno de los barrios más profundamente deprimidos ("La Mina") junto a una de las zonas con mayor desarrollo económico ("Diagonal Mar"). El Departamento de Computer Science, CS [2], recibió el encargo de impartir la asignatura de Informática del Q1. Los alumnos cursan los dos primeros años de asignaturas generales sin separación entre titulaciones; en un mismo grupo coexisten estudiantes de diferentes especialidades. En las Pruebas de Acceso a la Universidad, PAU, cada especialidad tiene su nota

*Los autores reconocen financiación de la Unión Europea (fondos FEDER), el Ministerio Español de Economía y Competitividad (MINECO) bajo los projectos de refs. TIN2017-29244-R, TIN2017-86727-C2-1-R y TIN2017-89244-R. También reconocen la financiación de la Generalitat de Catalunya, Agència de Gestió d'Ajuts Universitaris i de Recerca, bajo el proyecto 2017 SGR-786 (ALBCOM) y 2017 SGR-856 (MACDA).

[1] eebe.upc.edu/es
[2] cs.upc.edu

de corte y son dispares con lo que los estudiantes son muy heterogéneos. Para otoño de 2016 y del 2017 [3]:

Código	Titulación	2016	2017
31051	Ing. Biomédica	11.278	11.270
31052	Ing. de la Energía	7.928	8.209
31025	Ing. de Materiales	5.000	5.000
31053	Ing. Elect.	5.000	5.000
31054	Ing Elect. Ind. y Aut	9.000	8.228
31027	Ing Mecánica	7.482	7.688
31055	Ing. Química	5.000	5.000

El número de alumnos en el Q1 es de unos 700 y en el Q2 de aproximadamente 150. En otoño de 2017 hubo una reorganización de la docencia en la EEBE , que podujo un brusco y notable incremento en el número de profesores (18 docentes). Por ejemplo, en el Q1 de 2017, como los grupos de teoría son de un máximo de 60 alumnos, y se subdividen cuando hay alumnos suficientes en tres grupos de laboratorio de un máximo de 20 alumnos, fueron necesarios 12 grupos de teoría y 36 grupos de laboratorio. Para ello fue necesaria la colaboración de 18 profesores con diferentes niveles de dedicación a la asignatura. Además la docencia se compartió entre dos departamentos.

En este contexto, lo más fácil es que hubiera habido un caos enorme e incluso desacuerdos importantes entre profesores. Pero nada de eso ha pasado, los resultados académicos han sido aceptables, incluso buenos. Creemos que el éxito lo debemos al modelo de coordinación adoptado y que designaremos con el neologismo *ordocoordinación*[4]. La ordocoordinación, está basada en *orientaciones generales* y en la generación de un conjunto *mínimo* de reglas. Ha sido un caso claro de *learning by doing* [11] que los profesores han construido a lo largo de estos cuatrimestres. Ha sido una coordinación de abajo a arriba profundamente democrática y participativa. Esta aproximación implica que a veces hay que llegar a consensos sobre puntos concretos con una cierta rapidez. En estas situaciones, el correo electrónico es insustituible. Debido al reply-to-all se ha generado un buen numero de emails. En el Q1 de 2017, el número aproximado es de 350 por profesor[5], con lo que el total de emails de coordinación ha sido $350 \times 18 = 6300$. La ordocoordinación ha permitido: primero, la coordinación entre los grupos (incluyendo los exámenes de laboratorio) y segundo ha dado a los docentes un cierto grado de libertad pedagógica necesaria para tratar las especificidades de los distintos grupos de estudiantes.

La experiencia nos ha convencido de que esta aproximación es la adecuada para estos casos masivos y que por ello vale la pena describirlo con un cierto detalle. Hay tres aspectos que merecen ser destacados. Primero, se analiza cómo tratar la validación de programas (Sección 3). El modelo de coordinación flexible ha permitido coexistir e ir integrando paulatinamente dos puntos de vista distintos respecto a la validación. Segundo, veremos cómo se desarrollaron las orientaciones para el diseño de los exámenes de laboratorio (Sección 4). La ordocoordinación también ha permitido integrar sin problemas docencia en inglés (Sección 5). En la Sección 6 se analizan los resultados obtenidos. Finalmente en la Sección 7 se comentan las conclusiones de esta experiencia.

Como se trataba de impartir una nueva asignatura la ordocoordinación ha constado de dos partes inseparables; la coordinación de los profesores (Secciones 2, 4) y al, mismo tiempo, el diseño y evaluación a nivel docente (Secciones 2, 3, 5, 6). Bajo el punto de vista de los profesores ambos aspectos han sido indistinguibles y por ello aparecen conjuntamente en este artículo.

2. Programa

En Informática I se usa Pyhton como lenguaje de programación. Python permite el diseño de un curso basado en algoritmos más estructuras de datos. Seguidamente damos el esquema:

1. Variables, expresiones y tipos. `IDLE`.
2. Funciones, composición secuencial y alternativa. Doctest.
3. Composición iterativa (`for`, `range`), strings y no mutabilidad. Jutge.
4. Recorrido y búsqueda. Abstracciones lambda.
5. Listas homogéneas y heterogéneas.
6. Instrucción `while`. Problemas naturales.
7. Listas de listas como matrices o registros. Introducción al list comprehension.
8. Diccionarios: ejemplo lista de frecuencias. Introducción a los ficheros invertidos.
9. Diccionarios y listas: agenda.
10. Ficheros, opcional según calendario.
11. Preparación al examen escrito.

Al comenzar el Q1 de 2017 los puntos del programa y el lenguaje de programación estaban fijados[6]. Además el Coordinador había preparado una primera versión de ejercicios orientativos para cada sesión. Quedaba por

[3] `universitats.gencat.cat`

[4] La palabra *ordocoordinación* se ha inspirado en en la palabra *ordoliberalismo* que describe al liberalismo alemán que define un conjunto de reglas para permitir el desarrollo del mercado [5] (Cap 3).

[5] Quizá se podría utilizar `https://piazza.com/` para reducir los emails. Sin embargo hay que notar que los emails se generan debido a la necesidad de coordinación muy rápida entre docentes. Los emails entre docentes y estudiantes se gestionan por Atena, la plataforma Moodle de la UPC, `atenea.upc.edu/`.

[6] El *razonamiento computacional*, computational thinking, es el objetivo pedagógico básico. Sin embargo hay que desarrollarlo en un contexto prefijado. Las estructuras de datos forman parte del programa y no se pueden relegar a un (inexistente) segundo curso.

adaptar la propuesta a la realidad, ver la viabilidad global de la propuesta y (lo más importante) su recepción por parte de los estudiantes. A lo largo de estos cuatrimestres, los items se han modificado y la colección de ejercicios se ha enriquecido y adecuado[7].

Python, y la ordocoordinación, ha permitido introducir (aunque en este curso sea difícil profundizar en ellos) temas que no se tratan habitualmente en el primer curso de programación como las lambda-abstracciones, las list-comprehensions o los ficheros invertidos.

3. Validación automática

Uno de los objetivos del curso es que los estudiantes adquieran destreza escribiendo códigos correctos para problemas simples. Para lograrlo, el uso de herramientas de validación es inestimable. Analizamos dos de ellas, *Doctest*, *Jutge* y su uso combinado.

3.1. Doctests

Un doctest es un texto con la sintaxis de una secuencia de instrucciones en una shell de Python, acompañada de la salida esperada de cada una. Este texto, encapsulado entre triples comillas, formará parte de la *documentación* de las funciones.

A continuación se muestra un doctest para la función `interMult(m1,m2)` que calcula la intersección de multiconjuntos representados como listas ordenadas:

```
a) Intersección vacía
>>> interMult([1, 2, 5, 7], [3, 3])
[]

b) Intersección no vacía
>>> interMult([1, 2, 2, 5, 7], [2, 3, 3, 7])
[2, 7]

c) La intersección tiene repetidos
>>> interMult([1, 2, 2, 5, 7], [2, 2, 2, 7])
[2, 2, 7]
```

Como vemos los doctests pueden incorporar comentarios que permiten al estudiante entender cuál ha de ser el *comportamiento input/output* de los programas. Pueden entenderse como una especificación informal. Python dispone de un módulo que reconoce estos doctests, ejecuta automáticamente las instrucciones que contienen, comprueba cada resultado respecto del esperado y contabiliza el número de doctests superados.

Al principio del curso los alumnos tienen escritos los doctests de los problemas, pero luego se les pide que los desarrollen con el objetivo de que razonen sobre el programa *antes* de codificar su solución. Para hacer un análisis lo más exhaustivo posible y obtener una seguridad razonable de la correción de la futura solución, son claves tanto la cantidad como la calidad de los casos especificados. Este análisis previo del problema a menudo conduce a los alumnos a encontrar más rápidamente una solución correcta.

Los doctests también pueden verificar otras propiedades de los programas, como *comprobar que un parámetro no es alterado* en una llamada. En Python listas y diccionarios son mutables (alterables). Veamos un doctest para `maximo2(l)` que devuelve el segundo mayor valor de la lista `l`, pero sin modificar `l`:

```
>>> l = [2, 76, 3, 24, 17]
>>> maximo2(l)
17
>>> l == [2, 76, 3, 24, 17]
True
```

En *funciones que devuelven reales* se usan para *comprobar redondeos en los resultados*. Por ejemplo, para la función `invert(l)` que dada una lista de números $v_i \neq 0$ devuelve la lista de sus inversos $1/v_i$, definimos:

```
>>> l = invert([1.5, 2, 7.3])
>>> for v in l:
...     round(v, 4)
0.6667
0.5
0.137
```

Los doctests de las *funciones que devuelven diccionarios* aceptan *cualquier permutación de las clave/valor* comparando con ==. Para la función `freqCons(pal)` que devuelve un diccionario con las apariciones de las consonantes de `pal` definimos:

```
>>> d = freqCons('consonantes')
>>> d == {'t':1, 's':2', 'c':1, n':3}
True
```

Resumiendo, a lo largo del curso el estudiante adquirirá la habilidad de incorporar doctests adecuados como parte de la documentación de las funciones, y aprenderá a usarlos como una herramienta que le permite verificar automáticamente la corrección de sus soluciones durante todo su proceso de desarrollo. El futuro ingeniero necesitará este tipo de habilidades cuando tenga que desarrollar aplicaciones modulares de mayor escala en las que puedan intervenir varios programadores. Por lo tanto el doctest permite tratar, de modo simple, pedagógico y versátil variedad de aspectos relacionados con la corrección de los programas.

3.2. Jutge

Las sesiones de laboratorio cuentan con el respaldo de un entorno on-line de validación automática de programas. Esta herramienta, conocida como "Jutge"[8], se aloja en `www.jutge.org` y fue creada con un doble objetivo. El primero, consiste en proporcionar a los

[7] Sin duda esta colección de ejercicios puede considerarse un cierto tipo de *patrimonio* (quizá inmaterial) pero no por ello menos importante.

estudiantes un conjunto de problemas de dificultad graduada y un futuro veredicto de sus soluciones. Pese a que puede llegar a etiquetar como correcto un programa que no lo es, los falsos positivos son muy infrecuentes. El segundo objetivo es permitir al profesor una organización más eficiente del laboratorio. El profesor puede crear *cursos virtuales* que constan de varias listas de problemas, cada una de ellas acoje a problemas de una misma unidad temática. El Jutge permite al estudiante progresar de una forma autónoma. El profesor puede dedicar más tiempo a la resolución individualizada de dudas y puede hacer un seguimiento global usando servicios del Jutge.

Se ha preparado un curso virtual. Su construcción ha supuesto un esfuerzo no inferior a *cien horas de trabajo*. Hay que tener presente que, para dar de alta un ejercicio se requiere: un enunciado, unos juegos de pruebas públicos y privados (lo suficientemente exhaustivos para que no abunden los falsos positivos), y una solución que será usada como referencia. Dicho curso virtual consta de ocho listas como vemos en el siguiente cuadro:

Unidad temática	**Ejercicios**
Expresiones	3
Composición secuencial	2
Alternativa	9
Iteración (1) (for)	16
Listas	10
Iteración (2) (while)	5
Listas de listas	5
Diccionarios	10

3.3. Uso combinado de Doctest y Jutge

Muchos de los profesores de la asignatura usamos, o habíamos usado, Doctest y/o Jutge en otras asignaturas, pero no teníamos la experiencia en combinarlas. Al inicio se optó por el uso obligatorio de Doctest y opcional del Jutge. Sin embargo, hemos podido experimentar que el uso combinado de ambas herramientas ofrece ventajas didácticas. A continuación argumentamos nuestra apuesta y explicamos una metodología de trabajo que hemos experimentado. Recordemos brevemente los enfoques llamados de caja blanca/caja negra.

Las pruebas de *caja blanca* se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código. El programador escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores adecuados. Al estar basadas en una implementación concreta, si ésta se modifica, seguramente, habrá que modificar las pruebas. Por otro lado, dada una función específica, se pueden diseñar pruebas que demuestren que dicha función satisface los requisitos de input/output. Se dice que dichas pruebas actúan como si la función fuera una *caja negra* que consume las entradas y proporciona unas salidas, que deberán concordar con las esperadas, sin tener en cuenta el método concreto usado en el diseño de su código. Es decir, son independientes del código.

La realidad nos muestra que, habitualmente, los estudiantes tienden a diseñar el conjunto de pruebas siguiendo el enfoque de caja blanca. Una posible explicación es que, en general, tienen a esforzarse mucho más en conseguir que su pedazo de código funcione a toda costa que en hacer un análisis más fundamentado de los requisitos que debe satisfacer su solución. Asumiendo tal contexto, es importante tener en cuenta que, a pesar de que el enfoque caja blanca permite diseñar pruebas muy exhaustivas, el programador puede fallar en su intento si pasa por alto requisitos sobre el resultado, aunque garantice la prueba exhaustiva de todos los flujos de ejecución de su código. Aún peor, el programador puede estar falseando involuntariamente las pruebas si ha malentendido dichos requisitos. En resumen, aún asumiendo el esfuerzo del programador, este tipo de enfoque no tiene por qué ser ni correcto ni, en realidad, exhaustivo.

Ante tal situación, el caballo de batalla de los profesores de programación acostumbra a ser el de fomentar que los estudiantes se esfuercen en razonar sobre los problemas más allá de generar un pedazo de código que pase un cierto número de pruebas. Sin embargo, en asignaturas de iniciación y con un enfoque muy práctico dirigido a estudiantes de ingeniería, no sería realista asumir que los estudiantes van a entender los fundamentos que justifican la necesidad de diseñar pruebas independientes del código. En esta cuestión, hemos encontrado un gran aliado en el uso de Jutge pues enfrenta a los estudiantes con la "cruda realidad": *conseguir un código que funcione correctamente de acuerdo a los requisitos del problema requiere un esfuerzo de análisis del problema*.

Una situación habitual en el laboratorio es la de un estudiante convencido de haber probado su código usando un Doctest. Sin embargo, el Jutge lo rechaza. En tal situación, el estudiante se ve obligado a aceptar que necesita más esfuerzo del que había invertido. Podríamos decir que Doctest y Jutge juegan roles de "poli bueno" y "poli malo". Dado un problema del Jutge, proponemos la siguiente metodología:

1. Analizar la casuística asociada al problema y escribir una primera versión de Doctest que incluya las pruebas públicas que propone el Jutge.
2. Diseñar el código.
3. Depurar del código en base a los posibles errores detectados en la ejecución del Doctest.
4. Enviar al jutge. Volver a 1 si el código es rechazado.

Por último, añadir que los estudiantes perciben como un gran triunfo la aceptación final del Jutge, ésto dinamiza las sesiones de laboratorio.

4. Diseño de Evaluaciones

La evaluación ha evolucionado a lo largo de los tres cuatrimestres. Al principio incluía un trabajo final de curso. Dicho trabajo suponía una gran cantidad de esfuerzo y no quedaba clara su utilidad por lo que se ha eliminado por consenso. Damos la versión actual que consta de *tres exámenes prácticos de laboratorio* y *una prueba teórica escrita*.

Comencemos con los exámenes de laboratorio. Hay gran cantidad de grupos de laboratorio, en otoño de 2017 había 36. Como hay tres exámenes, en este cuatrimestre de otoño ha habido que preparar $36*3 = 108$ exámenes. La única manera es la ordocoordinación; cada profesor prepara los exámenes de los grupos de los que se ocupa. Ésto permite tener en cuenta las peculiaridades de dicho grupo (fiestas y puentes). A fin de no crear agravios comparativos los profesores han ido desarrollado progresivamente unas *orientaciones generales*. Las propuestas de enunciado se distribuyen por correo y se reciben propuestas de mejora.

Seguidamente describimos las reglas consensuadas. Vemos que, si bien centran el contenido del examen y cómo se distribuyen los puntos, hay total libertad en el diseño de los ejercicios concretos. De este modo se logra que los distintos exámenes sean de algún modo comparables y también tener en cuenta las especiales ideosincracias de los distintos grupos.

El *primer parcial* de laboratorio ha de constar de:

- Ejercicio de `if` (4 puntos). El profesor redacta el enunciado y prepara un `doctest` privado para evaluación El alumno es responsable de aportar un `doctest` en su solución, que se evalúa.
- Ejercicio de recorrido con `for` (5 puntos). El profesor redacta el enunciado incluyendo un `doctest` público y prepara un `doctest` privado para evaluación.
- Una expresión compleja (1 punto). No es pregunta aparte, se integra en el enunciado de alguna de las otras preguntas.

Damos un ejemplo en que se ha tomado la aproximación de *completa el siguiente programa* (fill the gap). Contar con un `doctest` simplifica la situación.

```
'''
Dado n>0, diseñar una función
medio_lleno(n) tal que:
>>> print(medio_lleno(4))
****
 ***
  **
   *
>>>
'''
def medio_lleno(n):
    '''
    >>> print(medio_lleno(1))
    *
    >>> print(medio_lleno(3))
    ***
     **
      *
    '''
    ...
if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose=True)
```

La norma que pide que al redactar un enunciado se incluya un `doctest` *público* y además se prepare un `doctest` *privado* para evaluación se aplica en adelante a todos los casos en que se pida diseñar un programa.

Los requerimientos para el *segundo parcial* son:

- Un ejercicio de búsqueda con `for` sobre una lista unidimensional (5 puntos). El enunciado incluye una función ya programada que el alumno tiene que llamar desde su función.
- Recorrido con `while` (5 puntos). Utilizar centinela o propiedad del último elemento, la longitud no es conocida.

Para el *tercer parcial* contendrá:

- Un problema de listas de listas
- Un problema de diccionarios
- Un tercer problema a escoger de entre las combinaciones sugeridas:
  - Una pregunta de diccionarios simples y otra de diccionarios con listas o listas de diccionarios.
  - Un recorrido sobre un diccionario existente y otra de creación de un diccionario.
  - Una pregunta de matrices y otra de listas heterogéneas.
  - Una búsqueda con listas o un `split` más un `filter`.

A fin de *obtener información global* de todos los grupos hay también una *prueba escrita* (la misma para todos los grupos). El conjunto de profesores prepara esta prueba. La utilización de GoogleDocs ha simplificado la tarea al tener un único documento de trabajo. Al principio se recogen propuestas, se detectan ausencias, se vetan preguntas y al final se vota y el coordinador tiene la última palabra. Posibles temas las los ejercicios para el examen son:

- ¿Por qué no acaba este bucle?
- ¿Qué devuelve esta función?
- Completar un algoritmo, o en general preguntas sobre un algoritmo ya escrito.
- Ejercicios de ficheros.

- ¿Este algoritmo es una búsqueda o un recorrido?
- Diseña un algoritmo para resolver el siguiente problema.

Como ejemplo damos el siguiente ejercicio. Convierte la siguiente función con `for` en una con `while`:

```
def f(lis):
  na = 0
  for i in range(len(lis)):
    if lis[i]=='a':
      na = na + 1
return na
```

La ordocoordinación ha permitido llevar a cabo de manera aceptable la carga de trabajo asociado a la evaluación continuada. Además dicha carga se ha repartido (automáticamente) de modo proporcional a la dedicación docente. Se podría pensar que este método da exámenes de dificultades muy distintas. Hay la impresión entre los docentes de que ésto no ha pasado. Como validación, todos los ejercicios de examen de todos los grupos y sus soluciones son visibles para todos los estudiantes tras el examen (via el moodle de la asignatura). Es razonable suponer que si los estudiantes hubiesen detectado casos muy claros de desigualdad se habrían quejado. Ésto no ha pasado.

5. Grupo en Inglés

Cada vez es más común impartir grupos en inglés, con el fin de internacionalizar los estudios y ofrecer a los propios estudiantes un valor añadido a su formación. En la EEBE (antigua EUETIB) se inició en el curso 2012-2013 una experiencia docente con el objetivo de impartir Grados de Ingeniería Industrial en inglés. El primer año se seleccionó un grupo de 20 estudiantes, con nota de acceso superior a la media, y un nivel de inglés al menos de B1. Los resultados tanto académicos como de satisfacción fueron excelentes [7]. Además, se incluyeron temas que normalmente no se llegan a abordar como los métodos avanzados de ordenación.

A partir del siguiente curso, el grupo pasó a tener el tamaño estándar de 60 alumnos en clases de teoría y 20-25 en clase de laboratorio. Progresivamente, se han ido incorporando más alumnos de intercambio. El perfil de los estudiantes ha evolucionado. Desde el perfil inicial de alumnos seleccionados por su nota de acceso a un perfil más heterogéneo, entre el que se cuentan alumnos que, sin ser de intercambio, prefieren cursar sus estudios en inglés.

En el grupo de docencia en inglés se empezó en este curso a probar en un grupo de laboratorio una metodología activa denominada *Just-In-Time Teaching* que combina por un lado el uso mixto del campus virtual y las clases presenciales con una atención individualizada. Se trata de una técnica que se ha mostrado eficaz para evitar abandonos prematuros de los estudiantes. En nuestro caso, proponemos actividades on-line de programación que los alumnos deben resolver antes de llegar a la siguiente clase de laboratorio. Estas actividades son corregidas in-situ por el profesor de prácticas durante la clase, y a la vez se proponen nuevos ejercicios al alumno que debe resolver, pero esta vez graduados de nivel en función de los errores cometidos en los ejercicios previos. De esta manera nos aseguramos de que el alumno disponga de una retroalimentación inmediata e individualizada. La resolución de estos ejercicios, además, cuenta como una nota de calidad para otorgar su nota final.

A pesar de las limitaciones de esta prueba piloto, los resultados, tal y como se describen en [6], han sido muy satisfactorios y prometedores. En particular, la metodología reduce el absentismo a las clases de laboratorio y mejora el rendimiento académico entre el tercio de alumnos con notas que no son ni las mejores ni las peores. Es decir, los resultados de la prueba piloto sugieren que entre los alumnos que están alrededor del aprobado (por arriba o abajo) hay una mejora notoria de calificaciones probablemente por una mejor motivación e implicación en la asignatura debido a un seguimiento individualizado.

En el curso 2017-18 hemos ampliado esta prueba a dos grupos más de laboratorio así como a un grupo de docencia no en inglés. A pesar de que la motivación sigue siendo elevada, y el absentismo ha disminuido, hemos detectado que los grupos eran muy heterogéneos. Los alumnos no se distribuyen de forma aleatoria por los distintos grupos de laboratorio, y los horarios eran muy favorables a uno de los grupos. Por esta razón, uno de los grupos tenía alumnos con mayores notas de corte lo que ha imposibilitado establecer una comparación cuantitativa. Las encuestas de satisfacción se mantienen elevadas, siendo relevante alguno de los comentarios en los que los alumnos indican que realizar ejercicios de forma continuada del nivel que se pide en los controles, así como disponer de más ejercicios en la plataforma virtual les ha sido de mucha ayuda.

Se ha constatado que para aquellos alumnos que siguen con dificultad, una atención individualizada les ayuda a superar la asignatura. Se comenta que las metodologías activas acostumbran a ser más eficaces en grupos reducidos o en cursos posteriores al curso inicial de la Universidad. En esta sección hemos visto cómo una metodología activa como Just-In-Time teaching mejora aspectos de la docencia y satisfacción tanto de estudiantes como del propio profesor. Se ha creado una dinámica de trabajo y confianza positiva y nos anima a continuar y ampliar la experiencia en próximos cursos.

6. Los límites del rendimiento

Dado que Informática es una asignatura de Q1 de primer año, tiene sentido intentar comparar el rendimiento de dicha asignatura con información histórica del rendimiento de los estudiantes antes de entrar en la universidad. Con todas las limitaciones necesarias tomamos como medida previa del rendimiento la notas de las PAU. Por lo tanto analizamos dos flujos, las notas de las PAU y las notas finales de Informática. Recordemos que catorce es la nota máxima de acceso a la universidad.

Dado un flujo de notas $x = (x_1, \ldots, x_n)$, donde x_i es la nota de las PAU del estudiante i, consideramos los siguientes indicadores: el máximo, max(x), la media (average, avg(x)) y la desviación estándar (standard deviation, st(x)) para medir la dispersión:

$$\mathsf{max}(x) = \mathtt{max}\{x_i \mid 1 \leq i \leq n\}$$
$$\mathsf{avg}(x) = \frac{1}{n}\sum_i x_i$$
$$\mathsf{st}(x) = \sqrt{\frac{1}{n-1}\sum (x_i - \mathsf{avg}(x))^2}$$

Quiza no tan conocido sea el Coeficiente de Gini [4, 3]. Dicho coeficiente se utiliza en ciencias sociales para medir el grado de desigualdad. Recientemente también se ha aplicado a la docencia[1, 2]. Formalmente, el *coeficiente de Gini* definido por:

$$\mathsf{gini}(x) = \frac{1}{\mathsf{avg}(x) \times n \times (n-1)} \sum_i \sum_{j>i} \mid x_i - x_j \mid$$

cumple $0 \leq \mathsf{gini}(x) \leq 1$, a diferencia de la medida usual de dispersión $\mathsf{st}(x)$. Cuando el gini es 0 tenemos a una sociedad muy igualitaria y cuando es 1 muy desigual. Tenemos:

Agregados					
	Q1	max	avg	st	gini
PAU	2016-17	12.88	9.18	1.68	0.1
	2017-18	13.31	9.16	1.72	0.1
Inf	2016-17	10	5.56	2.63	0.26
	2017-18	10	4.22	2.76	0.37

Vemos por ejemplo que las PAU son más regulares que Informática y que Informática es más desigual que los resultados de las PAU. En una sociedad con mucha desigualdad, la media es un mal descriptor de lo que sucede. Para ponderar entre si los efectos del promedio y la desigualdad, Amartya Sen [10] ha introducido la satisfacción $\mathsf{sat}(x) = \mathsf{avg}(x) \times (1 - \mathsf{gini}(x))$:

	Q1	sat
PAU	2016-17	8.26
	2017-18	8.24
Inf	2016-17	4.11
	2017-18	2.65

Sabemos que la satisfacción nunca supera a la media. Pensamos que la satisfacción captura mejor que la media el resultado de un curso. Después de todo tiene en cuenta cómo se reparte la media entre los estudiantes.

Seguidamente consideremos la *distribución del rendimiento* de los estudiantes. Una manera de hacerlo es cuantificarlos en tres grandes grupos: rendimiento alto, medio y bajo. Para ello adaptamos la división $(10\,\%, 40\,\%, 50\,\%)$ usada en economía para analizar la distribución de las rentas [9]. Diremos que el $10\,\%$ superior tiene *rendimiento alto* , el $40\,\%$ tiene *rendimiento medio* y al $50\,\%$ restante tiene *rendimiento bajo*. Disponemos de dos conjuntos de notas, las PAU y los resultados de Informática. En la siguiente tabla vemos las notas promedio para la división $(10\,\%, 40\,\%, 50\,\%)$ para los dos últimos cursos:

Promedio $(10\,\%, 40\,\%, 50\,\%)$				
	Q1	$10\,\%$	$40\,\%$	$50\,\%$
PAU	2016-17	11.97	10.21	7.82
	2017-18	11.97	10.22	7.77
Inf	2016-17	9.20	7.23	3.52
	2017-18	8.92	6.01	1.86

Vemos que la distribución no ha cambiado en las PAU. En Informática ha empeorado ligeramente a lo largo de estos dos años.

Puesto que tanto los resultados de las PAU como los resultados de Informática miden el rendimiento, es interesante considerar hasta qué punto son parecidos. Dado que el resultado de las PAU oscila entre $[0, 14]$ y el de informática entre $[0, 10]$ no son directamente comparables. Una manera de hacerlo es tomar $\mathsf{PAU}/14$ y $\mathsf{Inf}/10$. De este modo en ambos casos obtenemos números que miden el rendimiento entre $[0, 1]$:

Fracción $(10\,\%, 40\,\%, 50\,\%)$				
	Q1	$10\,\%$	$40\,\%$	$50\,\%$
$\frac{\mathsf{PAU}}{14}$	2016-17	0.85	0.72	0.55
	2017-18	0.85	0.73	0.55
$\frac{\mathsf{Inf}}{10}$	2016-17	0.92	0.72	0.35
	2017-18	0.89	0.6	0.18

Vemos que para el $10\,\%$ y el $40\,\%$ tenemos que $\mathsf{Inf}/10$ son parecidas a $\mathsf{PAU}/14$. De ser cierto, la nota de las PAU nos permitiría predecir aproximadamente la nota media para los alumnos de rendimiento alto y medio:

$$\mathsf{Inf} \approx \frac{10}{14} \times \mathsf{PAU}$$

Vemos que para el caso del $50\,\%$ podemos acotar :

$$\mathsf{Inf} < \frac{10}{14} \times \mathrm{mín}\{7.82, 7.77\} = 5.54$$

Parece que si las PAU predijesen, a niveles bajos correctamente, el número de aprobados debería ser mucho más alto. Por qué aparecen números bajos como $3{,}52$ o $1{,}86$ es algo que invita a la reflexión.

7. Conclusiones y Trabajo Futuro

La ordocoordinación ha permitido desarrollar y consolidar el curso de Informática I de la EEBE de manera satisfactoria en poco tiempo. Ha permitido integrar con éxito docentes de muy distintas trayectorias y desarrollar material sólido. Sería interesante una medida explícita, que no tenemos, del grado de satisfacción de los docentes. Los autores de este artículo creen que la experiencia ha sido muy satisfactoria.

Hay aspectos que aún requieren desarrollo. Hay que proseguir con la integración de `Doctest` con `Jutge.org`. Hay puntos de infraestructura que deben mejorar. A pesar de que la EEBE es un edificio nuevo, apenas hay enchufes en las clases de teoría. Ésto hace que les sea difícil a los estudiantes participar activamente con portátil. El nivel de seguridad informática durante los exámenes de laboratorio también necesita mejorar. Creemos que no tiene demasiado sentido pedirle a los profesores que desconecten el cable de conexión a Internet (de cada ordenador) antes del examen. La Escuela ya está dando pasos en la dirección adecuada para solucionar este problema. A diferencia de lo que ocurre en la Facultad de Informática de la UPC donde los exámenes de programación de algunas asignaturas se realizan en el entorno Jutge, la infraestructura del EEBE imposibilita este uso. A medio plazo y disponiendo de más recursos puede considerarse esta posibilidad (que cuenta con muchos partidarios, pero también algunos detractores, entre el profesorado).

No hay que olvidar que la programación (y su enseñanza) es un ser vivo. Es cierto que enseñamos a programar y para ello utilizamos un lenguaje de programación, en este caso Pyhton. El *punto conflictivo* está en determinar hasta qué punto el lenguaje de programación determina los contenidos del curso. Consideremos un ejemplo. A partir de *Orgullo y Prejuicio*, tenemos un diccionario en el que para cada señora aparecen ordenadamente sus amigos:

```
friends = {
 "lizzy":
   ["darcy", "bingley", "fitzwilliam",
     "wickham", "gardiner"],
  "lydia":
    ["wickham","fitzwilliam","darcy","bingley"],
   ...
}
```

Se pide diseñar `best_friends_of` tal que dado `friends` el nombre, `name`, de un caballero y un `k`, retorne (ordenadas lexicográficamente) la lista de señoras que tienen al caballero entre los `k` primeros amigos. Una posible solución es:

```
def best_friends_of (D, name, k):
    best_friends=[]
    for w in D:
        if (name in D[w][:k]):
           best_friends.append(w)
    return sorted(best_friends)
```

Quizás esta solución no sea la esperada por programadores entrenados en C++. Parece claro que al diseñar un curso de programación hay que contar con el lenguaje de programación, de lo contrario hay peligro de crear dificultades donde nos las hay. Parece claro que los items de la Sección 2 deberían poder enseñarse en cualquier lenguaje. No es fácil encontrar un equilibrio.

Referencias

[1] Maria J. Blesa, Amalia Duch, Joaquim Gabarro, Jordi Petit, and Maria Serna. Análisis de la evolución de un curso: Productividad y desigualdad. In *Actas de las XXII Jornadas de Enseñanza Universitaria de Informàtica, Jenui 2016*, pages 153–160, 2016.

[2] Maria J. Blesa, Amalia Duch, Joaquim Gabarro, Jordi Petit, and Maria Serna. Economía aplicada al estudio de la evolución de un curso. *ReVisión*, 10(1):55–70, 2017.

[3] L. Ceriani and P. Verme. The origins of the Gini index: extracts from Variabilità e Mutabilità (1912) by Corrado Gini. *The Journal of Economic Inequality*, 10(3):421–443, 2012.

[4] Corrado Gini. *Variabilità e Mutuabilità. Contributo allo Studio delle Distribuzioni e delle Relazioni Statistiche*. C. Cuppini, 1912.

[5] Christian Laval and Pierre Dardot. *La Nueva Razón del Mundo*. Gedisa, 2013.

[6] Antoni Perez-Poch and David López. Just-in-time teaching improves engagement and academic results among students at risk of failure in computer science fundamentals. In *Proceedings of the 47th ASEE-IEEE Frontiers in Education Conference*, 2017. doi:10.1109/FIE.2017.8190585.

[7] Antoni Perez-Poch, Fermín Sánchez, Nuria Salán, and David López. Análisis de rendimiento de un grupo piloto de Grado con docencia en inglés. In *Actas del Simposio/Taller XX Jornadas de Enseñanza Universitaria de Informàtica, Jenui 2014*, pages 19–25, 2014.

[8] Jordi Petit, Omar Giménez, and Salvador Roura. Jutge.org: an educational programming judge. In *43rd ACM Technical Symposium on Computer Science Education (SIGCSE)*, pages 445–450, 2012.

[9] T. Piketty. *El Capital en el Siglo XXI*. RBA, 2015.

[10] A. Sen. Real National Income. *The Review of Economic Studies*, 43(1):19–39, 1976.

[11] Robert Solow. *Learning from Learning by Doing*. Stanford, 1997.