ReVisión
Revista de Investigación en Docencia Universitaria de la Informática

AENUI Asociación de Enseñantes Universitarios de la Informática

Artículo invitado

De los bits a los modelos pasando por Bolonia

Jesús García Molina
Universidad de Murcia

Resumen

Con motivo de recibir el Premio AENUI a la Innovación Docente, en este artículo he repasado los principales retos a los que me he enfrentado en mi trayectoria docente y he comentado algunas lecciones aprendidas, con la intención de que algunas de las ideas que aquí se exponen puedan ser de utilidad a profesores noveles. También se incluye una corta reflexión sobre la implantación del proceso de Bolonia.

Palabras clave: Premio AENUI, iniciación a la programación, orientación a objetos, desarrollo dirigido por modelos, proceso de Bolonia, escuela de verano.

1. Introducción

Agradezco a los editores de ReVisión la invitación a escribir un artículo con motivo de la concesión del Premio AENUI a la Innovación Docente que me fue concedido en junio de 2016. Por diversas cuestiones personales, he ido atrasando su escritura y se da la circunstancia de que ReVisión ha publicado su último número sin tenerlo todavía preparado. Sólo la gentileza del editor y el hecho de tratarse una revista digital habrá permitido que usted esté leyendo estas líneas. Si así sucede, al agradecimiento por la invitación se unirá otro por aceptar fuera de plazo el artículo.

Cuando viajaba en autobús de Murcia a Almería (ciudad en la que se celebró JENUI 2016) para recoger el premio, pensaba en qué diría en la ceremonia de entrega y, por tanto, en las razones de AENUI para concedérmelo, lo que me llevó a pensar en qué había hecho en mis 33 años de docencia que fuese destacable para ser merecedor del reconocimiento de mis compañeros, más allá de intentar ser honesto en la tarea de enseñante, lo que entiendo significa preocuparse de enseñar lo que los alumnos deben aprender y hacerlo de un modo en el que los alumnos aprendan lo que deben aprender.

Supuse que el premio era motivado por un reconocimiento a mi trayectoria de 2002 a 2008 en el Comité Directivo de JENUI, aunque esto me despertó dudas al entender que otros compañeros de aquellos años lo merecían antes que yo. Me había unido a la "familia" de JENUI en 2001, cuando las Jornadas se celebraron en Palma de Mallorca, presentando una ponencia titulada «¿Es conveniente la Orientación a Objetos en un primer curso de programación?» Y desde entonces no dejé de asistir y participar hasta las JENUI de Granada. En todos esos años se sucedieron al frente del Comité Directivo Rosalía Peña, Cristóbal Pareja y Joaquín Ezpeleta y trabajamos en conseguir que JENUI fuese un punto de encuentro para todos los profesores universitarios de la informática con inquietudes en mejorar su trabajo docente y en presentar sus experiencias, al mismo tiempo que intentamos encontrar un nivel de calidad de las publicaciones apropiado para unas jornadas de la naturaleza de JENUI.

Las líneas que siguen son una versión extendida de lo que me hubiese gustado decir en la ceremonia de Almería y no pude hacerlo en los 5 minutos de los que dispuse (como es lógico, no estaba en el guion una perorata del premiado). En ellas, primero comentaré los principales retos a los que me he enfrentado en mi trayectoria docente, organizados en tres etapas. A continuación, presentaré algunas reflexiones sobre lo que ha sido el proceso de Bolonia en el que me impliqué pronto en mi condición de decano de la Facultad de Informática de la Universidad de Murcia (UMU) de 2000 a 2004. Y acabaré con unas pocas lecciones aprendidas destinadas a profesores noveles, aunque seguro que he podido olvidar alguna importante.

2. Una visión holística de la informática

Al acabar la licenciatura de Químicas, disfruté de una beca de un año de la empresa Bull para formar programadores COBOL. Al acabarla y sin ninguna experiencia docente obtuve una plaza de profesor Colaborador en la Escuela Universitaria de Informática de UMU en octubre de 1984. Se me asignó docencia de teoría en una asignatura de «informática general» y de prácticas con lenguaje Basic en una asignatura de «introducción a la programación». En aquellos años, esa asignatura de «informática general» era habitual en la mayoría de planes de estudios, aunque es cierto que la visión general variaba entre universidades. Y el primer tema de estas asignaturas solía ser un repaso a la historia de la informática. De modo que, en mi primera clase en la universidad, me encontré delante de unos 150 alumnos (había un grupo único en primer curso, Bolonia y los grupos reducidos quedaban lejos) hablando del ábaco, las calculadoras de Pascal y Leibniz, las máquinas de Babbage y de Ada Lovelace. Cuando recuerdo aquella primera clase, me sorprende que no estaba nada nervioso y sólo se me ocurre como explicación que «la ignorancia es muy atrevida». En el laboratorio no sólo tuve que lidiar con los GOTO sino con terminales conectados a través de una conexión telefónica de 9600 baudios con un Bull DPS 7. Una conexión muy lenta que provocaba gran tensión en los estudiantes, sobre todo en los exámenes. Esas prácticas no se pudieron realizar en PC hasta tres años más tarde cuando el centro pudo adquirir unos Olivetti M-19.

Para el curso siguiente recibí el encargo de hacerme responsable de la asignatura anual de informática general que incluía las prácticas en lenguaje ensamblador. Implanté un programa docente (ahora denominado guía docente) destinado a conseguir que los estudiantes consiguiesen una visión holística del ordenador a partir de la idea de *máquina multinivel* (organizada en varios niveles de abstracción), de modo que el alumno aprendiese cómo se llega a ejecutar un programa escrito en un lenguaje de programación. Empezábamos con circuitos digitales, luego estudiábamos la estructura y funcionamiento de un procesador y la entrada/salida, después las nociones básicas de sistemas operativos y acabábamos estudiando los fundamentos de los lenguajes de programación y la estructura básica de un compilador. Estas asignaturas de «informática general» han desaparecido de nuestros planes de estudio, aunque sigo creyendo que, en un primer curso, es conveniente ofrecer al alumno una visión global de la disciplina y de lo que subyace a la ejecución de un programa. En este sentido, recientemente, he sugerido la realización de un documental que ilustre de forma didáctica todo lo que sucede para que se llegue a ejecutar una búsqueda en un navegador.

Pronto comprendí que los programas de simulación podían ser muy útiles para el aprendizaje e incorporé a las prácticas «The visible computer» (TVC), un programa educacional que descubrí por casualidad y que simulaba el comportamiento de una CPU 8088 [1]. Además, en colaboración con dos estudiantes, desarrollamos un simulador de la unidad de control micro-programada de Tanenbaum [19] que se explicaba en la asignatura, primero en Pascal y luego en Smalltalk.

En esos primeros años adquirí conciencia del gran valor de los libros de texto y aprecié en gran medida el esfuerzo de aquellos que los escribían. Tuve la fortuna de disfrutar de la publicación de algunos excelentes libros que me ayudaron a preparar la asignatura, pero se publicaban en inglés por lo que era preciso escribir apuntes o libros para los alumnos hasta que se tradujesen o apareciesen textos en español. Recuerdo la impaciencia con la que esperaba la llegada de nuevos libros en los temas que me interesaban o la alegría con las traducciones al español de libros que podían ser útiles a los alumnos como los de Lister [13] o Tannenbaum [19]. Y dediqué esfuerzo a la elaboración de apuntes desde mi primer año como profesor. El material de prácticas dio lugar al primer libro del que he sido coautor, «Prácticas en Ensamblador 8086/88» [8], que abordaba el uso del simulador TVC y la programación en ensamblador 8086/8088. Escribiendo este artículo he sabido que ya entonces se había publicado en Estados Unidos un libro con el mismo objetivo, pero no tuvimos conocimiento de ello. En aquellos años, era algo que se te podía escapar fácilmente dado que libros de pequeñas editoriales no gozaban de buena difusión.

Hoy el mundo editorial ha sufrido una revolución con Internet y casi toda la información está al alcance de nuestros dedos en cualquier momento. En informática ya disponemos de buenos libros de texto para todas las materias básicas y, por ello, se presta poca atención a las novedades editoriales. Pasó el tiempo en que una vez al año recibíamos la visita del comercial de las editoriales universitarias, ahora nuestros estudiantes compran pocos libros, los canales de difusión son otros y existe Amazon.

No me gusta recordar de esos primeros años, el alto nivel de exigencia de los exámenes (incluían una parte test de cuatro respuestas y cualquiera podía ser verdadera o falsa) y el elevado número de alumnos que suspendían. Es evidente que no tenía la formación requerida para evaluar de forma proporcionada, algo que observo que es común en los profesores noveles que normalmente son más exigentes de lo debido.

3. Dándole vueltas a cómo enseñar a programar

Treinta años después de haberme enfrentado al reto de diseñar el programa docente de una asignatura anual de Introducción a la Programación (me referiré a estas asignaturas como IP y como IP1 a la parte correspondiente al primer cuatrimestre), soy testigo de dos hechos relacionados con la enseñanza de la programación que llaman mi atención, uno de ellos me alegra enormemente y el otro me causa cierta desazón.

Por una parte, observo un fenómeno que he anhelado desde que descubrí las posibilidades de la programación para edu-

car la mente de los jóvenes. Me refiero al creciente interés de que los alumnos de primaria y secundaria se inicien en la programación. Han surgido un gran número de asociaciones destinadas a fomentar la programación entre los jóvenes y cada vez son más los países que lo incluyen como formación curricular o plantean hacerlo a corto plazo. Parece que educadores y políticos (y padres en menor medida), al fin, se han dado cuenta de que la programación «educa la mente» o proporciona «alas para la mente» como sugería Horacio Reggini en el título de su libro de programación con Logo escrito en 1984 [17].

Cuando a finales de 1988 tuve que diseñar un programa docente para un primer curso de programación tenía dos ideas claras. La primera que Pascal era el lenguaje apropiado para expresar los algoritmos (hasta entonces se usaba Basic en mi centro). Esto parecía algo obvio, Pascal había sido diseñado por Niklaus Wirth con el objetivo de proporcionar la expresividad apropiada para enseñar a programar, y ya era muy utilizado en universidades de todo el mundo. Y la segunda idea era que no se podía reducir la enseñanza de la programación a simplemente enseñar un lenguaje de programación. La mayoría de universidades de todo el mundo seguían este segundo enfoque que era reflejado por la mayoría de libros de introducción a la programación de aquellos años. Estos libros tenían un título como «Introducción a la Programación con el lenguaje X» y describían el lenguaje X con ejemplos de programas que ilustraban la sintaxis y semántica de sus instrucciones, junto con una discusión de los algoritmos clásicos de un curso de IP. Sin embargo, yo no tenía clara la respuesta a la cuestión de cómo enseñar a programar, y comencé a buscar y estudiar material publicado (libros, artículos, recomendaciones curriculares) que ofreciese algún tipo de respuesta.

Dos textos fueron los que más influyeron en el enfoque que planteé hace tres décadas y que todavía se sigue en la asignatura de IP1 del Grado de Informática que se imparte en mi facultad, con variaciones que no afectan a la idea central. El que más influyó fue «Esquemas algorítmicos fundamentales: Secuencias e iteración» de Scholl y Peyrin [18], el cual reflejaba la experiencia de sus autores en el grupo de trabajo Anna Gram que a principios de los ochenta realizó un riguroso trabajo sobre cómo enseñar a programar [2]. Scholl y Peyrin presentaron un enfoque algorítmico para IP centrado en el uso de la noción de secuencia y el razonamiento inductivo para instruir a los alumnos en la construcción de iteraciones, algo esencial en un curso de iniciación a la programación.

«How to solve it» de Richard Dromey [6] fue el segundo libro que influyó en mi planteamiento. Su autor se inspiró en el trabajo de Polya sobre resolución de problemas y combinó técnicas de resolución de problemas con el diseño descendente para presentar un conjunto de estrategias para "descubrir" algoritmos eficientes y bien estructurados. Aplicó estas estrategias para mostrar cómo resolver problemas fundamentales en computación. La separación entre el diseño de un algoritmo y la implementación en un lenguaje (Pascal en este caso) es otra de las ideas centrales de su excelente libro. Merece la pena resaltar que Dromey afirmaba en el prólogo de este libro, que fue escrito en 1982, que los problemas de los estudiantes para aprender a programar no tenían que ver con aprender un lenguaje de programación sino con su falta de destrezas en los aspectos de resolución de problemas que son inherentes a la disciplina de la programación y sugería que esto se debía a que la escuela se centraba en enseñar a responder a preguntas y recordar hechos más que en enseñar a resolver problemas, una reflexión que sigue teniendo vigencia en la actualidad.

Las ideas de Scholl y Peyrin y de Dromey junto a otros trabajos influyentes como los de E.W. Dijkstra y N. Wirth sobre programación estructurada [5, 20], conformaron el enfoque semi-formal que finalmente planteé procurando integrar de forma coherente todas esas ideas. De nuevo, dediqué esfuerzos a la elaboración de un texto que apoyase el aprendizaje de los alumnos. En colaboración con los profesores que impartieron la asignatura cuando yo tuve que abandonarla, primero preparamos un texto-guía de la UMU que fue publicado en 1999 para la primera parte de la asignatura y más adelante abordamos un proyecto más ambicioso que acabó en 2005 con la publicación de un libro para un primer curso de programación completo [9]. La experiencia con este libro no ha sido buena por su escasa aceptación. A ello quizá haya contribuido que la editorial no realizó la difusión prometida por Latinoamérica, y en cierto sentido me siento en deuda con los otros autores ya que no aceptamos la oferta de Pearson al haberme comprometido verbalmente con Thomson-Paraninfo.

He revisado la web de algunos centros de informática de universidades españolas y con sorpresa he observado, según deduzco de sus guías docentes, que la mayoría sigue un enfoque clásico centrado en un lenguaje de programación, que suele ser un lenguaje orientado a objetos. Puede ser que al final de los estudios, sus egresados sean tan buenos o mejores programadores que los de mi centro, pero sigo pensando que ese no es el camino adecuado y me vienen a la mente los trabajos de Dromey y de Scholl y Peyrin. Hay que ver la programación como una disciplina de resolución de problemas en la que debemos dotar a los alumnos de estrategias y herramientas conceptuales para dominar la complejidad de idear algoritmos y programas.

Recuerdo que, en mi caso, los alumnos no escribían programas sino algoritmos durante las primeras 8 semanas de clase, entonces les enseñaba la correspondencia entre la notación algorítmica empleada y Pascal, y empezaban a codificar y ejecutar los algoritmos que antes habían escrito y probado sobre papel. No era partidario de la idea de Dijkstra de que los alumnos no escribiesen programas durante el primer semestre, pero sí consideré que era muy útil estar algunas semanas escribiendo algoritmos antes de programarlos, como una forma de hacerles ver que el punto clave es el razonamiento algorítmico. Esto era posible porque se trataba de una asignatura anual, es muy complicado hacerlo en una cuatrimestral. Además, formaba grupos de 30 alumnos y discutía con ellos en pizarra la resolución de ejercicios de programación en un juego de prueba y error, en el que ellos eran quienes planteaban soluciones que se discutían entre todos, como una forma de educar su mente en el pensamiento algorítmico. La inducción se reve-

ló como una herramienta poderosa para enseñarles a construir iteraciones. Estas clases estaban inspiradas por el espíritu de Dromey, Scholl, Peyrin, Wirth, Reggini y tantos otros buenos maestros de la programación.

4. La atracción por los objetos

A finales de 1986, Luis Cearra Zabala visitó nuestro centro como miembro del tribunal que había de juzgar la primera plaza de profesor funcionario (Titular de Escuela Universitaria) convocada por la UMU para un área de informática, en concreto Arquitectura de Ordenadores. Yo sabía que él había traducido al español un conocido libro de programación en Pascal [7]. En algún momento de su corta visita, entró a mi despacho a saludarme y le invité a sentarse, Luis me preguntó a qué me dedicaba y yo le respondí con un «ando mirando muchas cosas en busca de una línea de investigación, ¿qué me sugieres tú?», el me respondió escuetamente «Smalltalk, los objetos». Yo no tenía ninguna idea sobre la programación orientada a objetos y asociaba Smalltalk a crear interfaces de usuario gráficas muy bonitas. Nunca he sabido si la recomendación de Luis sólo se refería a programar con Smalltalk o ya conocía (o intuía) el creciente interés por la orientación a objetos dentro de la comunidad del *software*. En 1991, asistí en Madrid a un seminario de Bertrand Meyer en el que expuso su visión del desarrollo de *software* orientado a objetos a través de su lenguaje Eiffel, lo cual había reflejado en su libro «Object-Oriented Software Construction» que había sido publicado recientemente [14]. Fue en ese seminario cuando empecé a vislumbrar las posibilidades del paradigma orientado a objetos en la creación de *software*.

Cuando en el curso 1991/92 arrancaron los estudios de Licenciatura de Informática en la recién implantada Facultad de Informática de la UMU, se me asignó un cuatrimestre (equivalente a unos 4,5 créditos actuales) de una asignatura anual de Ingeniería del Software. Aunque en las directrices del BOE no se hablaba nada de programación orientada a objetos, decidí dedicar la parte que me correspondía impartir a enseñar única y exclusivamente ese nuevo paradigma de programación, utilizando el libro de Meyer mencionado antes para la teoría y Smalltalk como lenguaje para abordar un pequeño proyecto de programación. Nos convertimos en el primer centro universitario español, y uno de los primeros del mundo, en ofertar una asignatura de programación orientada a objetos (eso sí, sin seguir las directrices BOE), lo que se ha mantenido de forma ininterrumpida hasta la fecha. Hoy ya es habitual que una formación en orientación a objetos se incluya en los planes de estudio del grado de informática de todas las universidades del mundo. Más tarde, sobre todo a partir de la aparición de Java en 1995, se produjo la adopción industrial de las tecnologías orientadas a objetos, que pasaron a ser las más usadas para crear nuevas aplicaciones. Cuando esto sucedió, nuestro centro ya ofrecía una formación "madura" en orientación a objetos que incluía también la enseñanza de modelado en UML y patrones de diseño.

Cabe destacar que Bertrand Meyer fue ponente invitado de las JENUI 2004, celebradas en Alicante, en las que planteó su enfoque para la iniciación a la programación con el lenguaje Eiffel y participó activamente en todas las sesiones, incluyendo la sesión en la que presenté una ponencia sobre el enfoque comentado arriba para un primer curso de programación y en la que se defendía la conveniencia de usar un lenguaje procedural.

A mediados de los noventa inicié una colaboración con Andrés Otero, editor de Prentice-Hall y Addison-Wesley, para asesorarle sobre qué traducciones de libros relacionados con la orientación a objetos podrían ser interesantes, y fruto de esa relación supervisé la traducción de libros de gran impacto como la segunda edición del libro de Bertrand Meyer mencionado arriba [15], el libro de UML de sus tres creadores [3] y el también exítoso texto de Craig Larman sobre UML y patrones de diseño [12].

En el año 2010, como parte de la celebración del XXV aniversario de nuestra facultad, la Universidad de Murcia concedió un Doctorado *Honoris Causa* a Alan Kay que había recibido el premio Turing por sus contribuciones a la orientación a objetos. Casi veinte años después de ser pioneros en la enseñanza de la programación orientada a objetos pudimos tener entre nosotros y conceder la máxima distinción de una universidad a la persona que creó Smalltalk y concibió los principios de ese paradigma de programación.

5. Del dirigido-por-objetos al dirigido-por modelos

Y tras la adopción industrial de la programación orientada a objetos, la comunidad del *software* dirigió su vista hacia el uso de modelos como artefactos que pueden "dirigir" de forma sistemática el desarrollo de *software* y posibilitan la generación automática de código; los modelos son creados con lenguajes de modelado o lenguajes específicos del dominio (DSL). De este modo, a principios de este siglo comenzó a emerger una nueva disciplina de la ingeniería del *software* que estudia todo lo relacionado con el uso de modelos de *software*: la *Ingeniería de Software Dirigida por Modelos* (*Model-driven Software Engineering*). Nuestro centro ha sido también pionero en la enseñanza de las técnicas básicas de esta nueva disciplina: metamodelado, creación de DSL y transformaciones de modelos. Primero fue a través de una asignatura de doctorado que arrancamos en el curso 2004/2005 y desde el curso 2006/2007 en una asignatura de máster de 6 créditos. Como me sucedió con la programación orientada a objetos, de nuevo he tenido la fortuna de que coincidiese la docencia con la investigación en la que estoy involucrado. Y de nuevo he impulsado y participado, como miembro de la Red Temática Nacional de «Desarrollo de Software Dirigido por Modelos», en la elaboración de un libro básico sobre los principios, métodos y técnicas de la *Ingeniería Dirigida por Modelos*, el cual está principalmente destinado a apoyar la docencia de

esta materia en las universidades españolas [10]. Ahora, mi gran sueño es disponer de un semestre sabático para escribir un texto que presente el desarrollo basado en modelos desde un punto de vista muy práctico, a partir de la experiencia acumulada en la asignatura de máster mencionada, en la que los alumnos se enfrentan a un proyecto abordándolo en cuatro entregables. Honestamente, creo que los grados debería incluir una formación optativa en Ingeniería Dirigida por Modelos que completase la formación en modelado y lenguajes de programación que es obligatoria.

Por otra parte, con la implantación del nuevo plan de estudios conforme a las directrices del denominado Proceso de Convergencia o Proceso de Bolonia, he organizado junto a dos compañeros una asignatura de tercer curso de grado destinada a ofrecer una formación básica en principios y tecnologías de desarrollo de *software*, en las que los alumnos deben enfrentarse en grupos de dos al desarrollo de su primera aplicación (este curso ha sido una aplicación con la funcionalidad básica de un servicio de reproducción de música como Spotify). Los alumnos aprenden patrones de diseño y código, la creación y uso de componentes *software*, una técnica para almacenamiento de objetos, el manejo de una librería de creación de GUI, el manejo de API externas, herramientas como Maven y Subversion para la gestión del proyecto, entre otros conocimientos y habilidades. Tengo pendiente escribir para JENUI una ponencia sobre esta asignatura cuyo planteamiento tiene algunos elementos originales que podrían ser de interés para la comunidad JENUI interesada en cómo introducir a nuestros alumnos de grado en la producción de *software*.

Por tanto, todavía sigo implicado en asignaturas en las que debo "tocar" *software* y herramientas (no tanto como lo que quisiera o debiera), algo que creo es fundamental para un docente que enseña materias relacionadas con el desarrollo de *software*.

6. La desilusión de Bolonia

La primera reunión de la Conferencia de Decanos y Directores de Escuela de Informática a la que asistí fue en febrero de 2000 en Las Palmas de Gran Canaria y la primera "tarea" a la que me enfrenté fue a pensar en un nombre que la designase. Allí estábamos todos los decanos/directores proponiendo nombres y escribí en un papel «CODI», me parecía que sonaba bien y me recordaba a Cobi, la mascota de los Juegos Olímpicos de Barcelona. Emilio Torrano, decano de la Facultad de Informática de la UPM, que estaba sentado a mi lado me dijo «Murciano, di ese nombre» (él también era murciano) y lo propuse, se añadió una «D» y surgió entonces el acrónimo de CODDI. Y fue en la CODDI donde empecé a tener contacto con el proceso de creación de un Espacio Europeo de Educación Superior (EEES) conocido como «proceso de Bolonia». Entendía que significaba la oportunidad de realizar una transformación de la universidad española para mejorar la docencia, y así lo señalé en algunas mesas redondas y charlas en las que tuve oportunidad de participar. Y por eso ha sido grande la desilusión con el resultado final, aun reconociendo que se han producido mejoras en algunos aspectos docentes.

Como es bien sabido, la idea de una armonización o convergencia en los estudios superiores de la UE surgió fundamentalmente por dos razones: facilitar la movilidad de estudiantes y profesores en los países de la UE y competir con Estados Unidos en la atracción de estudiantes dado que se trata de un mercado que mueve mucho dinero. La armonización consistía en cuatro medidas administrativas: organización de los estudios universitarios en tres ciclos (grado, máster y doctorado), definición del crédito europeo ECTS para medir el trabajo del estudiante, el suplemento al título para unificar la información sobre el título académico expedido al estudiante, y la creación de agencias de calidad nacionales para la acreditación de títulos.

En los documentos que se fueron difundiendo por la UE, estas cuatro medidas también se acompañaron con ideas relacionadas con la mejora de la calidad docente, lo cual en España generó ilusión en buena parte de la comunidad universitaria. El proceso de Bolonia abría la posibilidad de mejorar los planes de estudio, así como los métodos de enseñanza. Sin embargo, la incompetencia de los dirigentes políticos, María Jesús San Segundo y Mercedes Cabrera fueron las ministras implicadas, y la complicidad de una universidad principalmente preocupada por intereses espurios malograron la oportunidad. Lo que podría haber supuesto una renovación de la Universidad Española, ha provocado, en cambio, que viva uno de sus peores momentos. La mezcla de una mala gestión de la implantación del EEES unido a una crisis económica en el mundo occidental de un orden de magnitud similar al *crack* de 1929, ha provocado que la Universidad esté atravesando momentos muy complicados que costará mucho superar. Está fuera del ámbito de este artículo un análisis detallado sobre la implantación del EEES y sólo me limitaré a señalar algunas de las decisiones que, en mi modesta opinión, más han dañado a nuestro sistema universitario.

El Ministerio de Educación optó por una duración de 4 años para todos los grados en vez de favorecer la flexibilidad. Esta decisión ha tenido un alto coste para el país. Titulaciones que venían formando adecuadamente a profesionales en 3 años pasaron a tener un año adicional que poco aporta y que supone un elevado coste para las familias y universidades. Las familias deben pagar un año más de estudios y las universidades tuvieron que contratar más profesorado para afrontar la nueva docencia. La llamada Ley Wert modificó la normativa para permitir grados de 3 y 4 años, pero entiendo que ahora es difícil la vuelta atrás. ¿Cómo se obliga en las universidades públicas a «volver atrás» a las titulaciones que pasaron de 3 a 4 años con centros que dejaron de ser «escuelas» para convertirse en «facultades» y tener estudios de doctorado?

En el caso del Grado de Informática, como miembro del Grupo Ponente que elaboró el Libro Blanco, lideré junto a otros colegas una iniciativa para mantener el sistema 3+2. Nuestra propuesta contó con el apoyo de importantes universidades como todas las de Barcelona menos la UPC, Complutense de Madrid, Deusto, Girona, Illes Balears, Pontificia de

Comillas, UNED, Valladolid y Murcia, pero no resultó vencedora en la votación final. Recuerdo a un director de una Escuela Técnica de Informática que en una reunión de la CODDI me dijo: «Jesús, llevas razón en tu propuesta de 3+2, pero yo defiendo los intereses de una Escuela Técnica y el grado de 4 años nos beneficia mucho como es fácil de comprender, creceremos y tendremos el estatus de Escuela Superior». Esto es anteponer intereses particulares a los generales, en este caso de una titulación y de un país. Intereses particulares que, con frecuencia, dirigen la toma de decisiones en las universidades e instituciones públicas.

El Proceso de Bolonia promovía una enseñanza que daba más protagonismo al estudiante con menos clases presenciales y grupos más reducidos, más seminarios y tutorías con el profesor y también una formación más práctica. Todo esto podría suponer una mejora de la calidad docente. Sin embargo, resulta evidente, que esto requería una planificación plurianual con una memoria económica. Sin embargo, se aplicó todo a la vez con la implantación de los nuevos planes de estudio, lo cual exigió de la contratación de muchos profesores en aquellas titulaciones que anteriormente tenían poca carga práctica, no incluían la realización de trabajos fin de carrera, y sus grupos eran numerosos, como era el caso de Derecho o Educación. Ahora, las universidades se enfrentan al problema del excesivo número de profesores asociados, contratados como mano de obra barata. Un auténtico despropósito.

Otro despropósito ha sido el denominado Máster de Secundaria creado con el objetivo de formar a los titulados que aspiran a ser profesores de Secundaria y Bachillerato. Sólo una pregunta, ¿conocen a algún alumno que lo haya cursado o docente que no sea de las áreas de Educación que opine bien de este máster? Una de las conclusiones de un estudio reciente [4] es que «los alumnos opinan que [el máster] contribuye poco o casi nada a su formación, y solo consideran positivas las informaciones que reciben sobre el funcionamiento de los centros de enseñanza, algo que allí no constatan porque solo dan clases o ayudan al tutor».

A lo largo del artículo no he utilizado terminología propia de los métodos asociados al EEES, sólo en este punto señalaré el hartazgo que puede provocar lo que llamo «ingeniería de competencias», ese ridículo juego al que jugamos al rellenar guías docentes o al elaborar planes de estudio (me vienen a la cabeza las famosas «fichas de informática» y me irrito por su estupidez). Acabo de rellenar una ficha sobre la asignatura del máster que imparto en la que he tenido que puntuar cuántos créditos de la asignatura corresponden a una serie de competencias generalistas del plan de estudios. ¡Dios mío, cómo se calcula eso! Jamás me he quejado de que ahora exista más burocracia en grado y máster (sí en el doctorado, realmente me exaspera por su inutilidad) y creo que la idea de «competencia» puede ser muy útil, pero se ha llegado con ella a planteamientos sin sentido.

También es justo reconocer que el proceso de Bolonia ha motivado que en la mayoría de centros haya crecido el interés por las buenas prácticas docentes y se hayan establecido mayores controles para velar por la calidad. Estoy pensando en las labores de coordinación entre asignaturas, en el interés en conseguir que los alumnos adquieran las competencias transversales, en los sistemas de garantía de calidad, la aceptación de las guías docentes como contrato entre profesor y alumno (siempre debió ser así), por señalar algunas que considero más relevantes.

7. Algunas lecciones aprendidas

Me gusta la docencia, me gusta enseñar y ver que los estudiantes aprenden y que valoran que es útil lo que aprenden. Enseñar es un arte y como todo arte algunos tienen un don innato para que la obra sea de calidad y otros no tienen ese don o no en la medida deseada, pero pueden adquirir la destreza apropiada para conseguir ser buenos docentes. Por esto es tan importante que la universidad dedique recursos a la formación de profesores noveles, formación en la que deben participar profesores de las áreas relacionadas con la docencia a impartir, por lo que es muy importante la implicación de la facultad a la que pertenezca el docente.

Como en cualquier equipo humano que se enfrenta a un reto, la motivación es esencial y el docente debe tener la habilidad de saber motivar a los alumnos. En un primer curso de programación, los alumnos del siglo XXI esperan desde un principio aprender directamente a crear programas que produzcan bonitos efectos visuales, se aburrirán escribiendo programas para contar el número de ocurrencias de una palabra en un texto, además escritos en un viejo lenguaje como Pascal, pero un profesor que sepa motivarlos desde el primer día podrá conseguir que disfruten con esos ejercicios, es cuestión de decir las palabras de motivación adecuadas en cada momento.

Un profesor debe tener en mente que no estamos formando a alumnos que sean elegidos por Google como brillantes programadores, sino que la formación debe contemplar a alumnos con diversas capacidades. Y con esa perspectiva es preciso plantear la enseñanza y la evaluación.

Qué importante medir bien la carga lectiva de una asignatura y de un plan de estudios. Los días tienen sólo 24 horas, y un estudiante debería dedicar en promedio 40 horas semanales al estudio. Pero cometemos verdaderas barbaridades. Mientras escribo, no puedo evitar pensar que ahora mismo imparto una asignatura de 6 créditos en el plan de estudios pero que debía ser de 7,5 para satisfacer los objetivos (huyo de la palabra «competencia») expuestos en el BOE, en particular conseguir que el alumno se enfrente a su primer proyecto de programación que involucre una interfaz de usuario y almacenamiento externo. Los alumnos valoran muy positivamente la asignatura y el trabajo de los profesores, pero esta asignatura es un ejemplo de lo que no debe suceder nunca, aunque pueda “soportarse” que sea una única asignatura del plan de estudios.

El docente de informática tiene un trabajo extra con el cambio continuo que se produce en nuestra disciplina. Un profesor de derecho o filología clásica va acumulando cada vez más saber, uno de informática puede ver disminuir sus cono-

cimientos cada año que pasa. Pensemos, por ejemplo, en un profesor que pertenece al grupo docente de bases de datos, y lleva tiempo involucrado en docencia relacionada con sistemas relacionales, nos podemos preguntar si ese profesor puede ignorar la emergencia de las bases de datos NoSQL o de las tecnologías de almacenamiento para Big Data. No, en absoluto. Y deberá reflexionar si hay que introducir de inmediato en los planes de estudios del Grado una formación sobre sistemas NoSQL, en qué modo, si tiene sentido reemplazar parte de la enseñanza de los sistemas relacionales por los nuevos sistemas, entre otras reflexiones. En definitiva, siempre debemos estar atentos a la evolución del área en la que se enmarca nuestra docencia y reflexionar sobre qué debemos enseñar y cómo hacerlo. Y en este sentido, no se debe tener miedo al cambio, el cual muchas veces se debe a una cuestión de comodidad personal. Ahora mismo me veo involucrado ante una decisión de este tipo. En la asignatura sobre tecnologías de desarrollo de *software* utilizamos Subversión (SVN) como herramienta de control de versiones, dado que era el sistema más usado en 2009 cuando diseñamos la asignatura. Sin embargo, la adopción de Git va creciendo y ya supera a SVN. El cambio de SVN por Git significaría tirar a la basura el material preparado y tener que conocer en detalle Git. Pero, aunque nos resulté una incomodidad, debemos planteárnoslo y cambiar si creemos que Git es mejor para los intereses de los alumnos.

Las universidades se enfrentan al reto de combinar clases presenciales con enseñanza en línea. Esto requiere disponer de servicios que apoyen a los docentes en la producción de contenidos digitales. No obstante, los docentes, y más en el ámbito de la informática, ya deberían tener un gran interés por crear estos contenidos en línea de aprendizaje. Recuerdo una ponencia del colega Alberto Prieto en las JENUI 2016 de Almería [16] que me impresionó al ver como un profesor cercano a la jubilación había sido capaz de crear y llevar a la práctica unos magníficos contenidos digitales. Hoy no tenemos excusa para no tener grabadas todas nuestras clases para que estén a disposición de los alumnos y las puedan ver cuántas veces deseen. No quiere decir este comentario que piense que los buenos contenidos se reducen a tener clases grabadas. Estoy convencido de que los contenidos digitales no sustituirán a los buenos libros de texto, que seguirán siendo necesarios y animo a los buenos docentes a escribir buenos libros sobre su materia.

Realmente, cuando comparo la docencia de la universidad española de mediados de los ochenta con la actual, observo una mejora sustancial en todos los aspectos, por ello creo que debemos ser optimistas, aunque sin dejar de ser críticos con nuestro trabajo.

Y acabaré comentando la iniciativa que he impulsado de la que, probablemente, me siento más satisfecho: La «Escuela de Verano de Informática» que organiza desde hace 8 años mi facultad. Fue algo que se me ocurrió leyendo el libro «Una universidad para niños» [11] que había comprado a mis hijas. Este libro cuenta la experiencia de la Universidad de Tubinga (Alemania) que organizó charlas para niños en edad escolar sobre temas científicos y que fueron impartidas por investigadores expertos en el tema. De ahí surgió la idea de organizar un conjunto de seminarios sobre temas de informática para estudiantes de ESO y Bachillerato, que se celebrarían a lo largo de dos semanas, incluirían teoría y prácticas, y serían impartidos por profesores de nuestra facultad. La Escuela ha llegado a la novena edición y cada año tenemos una demanda muy superior a las 70 plazas ofertadas. Pero los detalles concretos sobre esta experiencia los dejamos para una ponencia en una próxima edición de JENUI, algo que tengo en mente desde hace varios años.

Referencias

[1] Charles Anderson. *The visible computer:8088: Assembly language teaching system*. IBM PC, Software Masters. 1985.

[2] Anna-Gram. *Raissoner pour programmer*. Dunod. 1986.

[3] Grady Booch, James Rumbaugh, e Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1996 (edición en español de 1999).

[4] Javier Cachón Zagalaz, Inés López Manrique, Santiago Romero Granados, María Luisa Zagalaz Sánchez y Carmen González González de Mesa. *Opinión de docentes y estudiantes del máster de secundaria sobre las aportaciones de este a la formación del profesorado, la calidad docente y los intereses personales*. Magister, Vol. 27. Núm. 1. Enero-Junio, 2015.

[5] Edsger W. Dijkstra. *Notes on Structured programming*, en O.J. Dahl, E.W. Dijkstra y C.A.R. Hoare, editores, *Structured programming*, Academic Press. 1972.

[6] Richard Dromey. *How to solve it by computer?*. Prentice-Hall. 1982.

[7] Willian Findlay y David A. Watt. *Pascal. An Introduction to Methodical Programming*, segunda edición. Pitman. 1984 (edición en español de Editorial Rueda, 1984).

[8] Jesús J. García Molina, Antonio F. Gómez Skarmeta, y José Pío Martínez Núñez. *Prácticas de ensamblador 8086/88*. Editorial PPU. 1990.

[9] Jesús García Molina, Francisco J. Montoya Dato, José L. Fernández Alemán, y María J. Majado Rosales. *Una introducción a la programación. Un enfoque algorítmico*. Thomson. 2005.

[10] Jesús García-Molina, Félix Garcia Rubio, Vicente Pelechano, Antonio Vallecillo, Juan Manuel Vara, y Cristina Vicente Chicote. *Desarrollo de Software Dirigido por Modelos*. RaMa. 2013.

[11] Ulrich Janssen y Ulla Steuernagel. *Una universidad para los niños*. Critica. 2003.

[12] Craig Larman. *Applying UML and Patterns*, segunda edición. Prentice-Hall. 2002 (edición en español de 2003).

[13] Andrew Lister. *Fundamentals of Operating Systems*, segunda edición. Springer-Verlag. 1979 (edición en español de Editorial Gustavo Gili, 1986).

[14] Bertrand Meyer. *Object-Oriented Software Construction*, primera edición. Prentice-Hall. 1988.

[15] Bertrand Meyer. *Object-Oriented Software Construction*, segunda edición. Prentice-Hall. 1997 (edición en español de 1999).

[16] Alberto Prieto Espinosa, Beatriz Prieto Campos, Begoña del Pino Prieto. *Una experiencia de flipped classroom*. Actas de las Jenui, vol. 1. 2016. Disponible en `http://www.aenui.net/ojs/index.php?journal=actas_jenui&page=article&op=view&path%5B%5D=268`.

[17] Horacio C. Reggini. *Alas para la mente: LOGO: Un lenguaje de computadoras y un estilo de pensar*. Ediciones Galápago. 1984.

[18] Pierre C. Scholl y Jean P. Peyrin. *Schémas algorithmiques fondamentaux: Séquences et itérations*. Masson. 1988 (Edición Española de 1991).

[19] Andrew Tanenbaum. *Structured Computer Organization*, segunda edición. Prentice-Hall. 1984 (edición en español de 1985).

[20] Niklaus Wirth. *Systematic programming: An Introduction*. Prentice-Hall.1973 (edición en español de Editorial El Ateneo, 1982).

Jesús García Molina es profesor Catedrático de Universidad del departamento de Informática y Sistemas de la Univesidad de Murcia. Dirige el grupo de investigación ModelUM cuyas líneas de investigación se centran en la aplicación de la ingeniería del software dirigida por modelos en áreas como la ingeniería inversa y la creación de contenidos digitales. En DBLP se pueden encontrar sus principales publicaciones en esas lineas. Es autor de varias publicaciones docentes (libros, artículos, informes) relacionadas con la programación e ingeniería del software. Recibió en 2016 el Premio AENUI a la Calidad e Innovación Docente. Miembro del Comité Directivo de JENUI de 2002 a 2008. Para más detalles puede enviarle un correo electrónico a jmolina@um.es.