Cómo NO hacer unas prácticas de programación

Agustín Cernuda del Río

Departamento de Informática - Universidad de Oviedo
Escuela Universitaria de Ingeniería Técnica en Informática de Oviedo
Facultad de Ciencias - C/ Calvo Sotelo, S/N - 33007 Oviedo
guti@lsi.uniovi.es

Resumen

En la enseñanza universitaria de asignaturas de programación los alumnos reproducen, invariablemente, multitud de actitudes altamente nocivas. Abarcan aspectos muy variados, tanto técnicos (codificación apresurada y desordenada, despreciar los mensajes de error del compilador) como de comportamiento (no acudir a tutorías o hacerlo sólo en las fechas previas a la entrega de una práctica) como de pura deontología (copia de prácticas). A pesar de las advertencias en contra, resulta extremadamente difícil erradicar estos hábitos.

Adoptando un enfoque alternativo, quizás el uso del humor y la ironía permitan que el mensaje llegue al alumno. Además, también puede ser interesante plantearse la enseñanza *en negativo*.

1 Introducción

La comunidad de alumnos resulta con frecuencia especialmente refractaria a cierto tipo de consejos sobre la programación y la realización de prácticas, cosa perfectamente lógica si se piensa en su perfil y su juventud. No hay que olvidar, además, que muchos profesionales (incluyendo los profesores y al autor de este artículo) caen en ciertos errores una y otra vez, con lo que casi no cabe culpar al alumno (más inexperto) por cometer esos mismos errores.

¿Qué se puede hacer al respecto? Está claro que de alguna manera hay que llamar la atención del alumno sobre su propio comportamiento y predisponerlo en contra de estas actitudes. Pero los consejos en positivo no suelen funcionar. Por eso hemos decidido adoptar otro enfoque: aconsejar (de manera irónica) lo peor. Esperábamos que esto ayudase por tres razones:

- El humor, la ironía, la ridiculización, resultan mucho más amenos para un alumno típico que los consejos académicos. Cabe esperar una mayor penetración del mensaje.
- Esta ridiculización *de comportamientos* (nunca de personas) puede ser una "vacuna" eficaz; el alumno estará sensibilizado contra lo absurdo de ciertas actitudes, y esto es un poderoso factor de motivación. Si el miedo al ridículo impide a los alumnos hacer preguntas, quizás les impida cometer errores[1].
- Hay ciertas habilidades que se pueden caracterizar en gran medida a partir de lo que *no* hay que hacer (por ejemplo, cómo *no* utilizar un procesador de textos [1]). En programación, una vez que se conocen las habilidades básicas, un buen programador se distingue en gran parte por las cosas que *no* hace.

Partiendo de estas premisas, decidimos redactar un documento que recogiese los errores más frecuentes, con el fin de que un alumno pudiese identificarlos con facilidad incluso cuando los cometiese de manera inconsciente, y publicarlo en forma de página web.

---

[1] Queda claro que esto puede aplicarse sólo con sumo cuidado y a errores evitables de actitud, no técnicos o de comprensión; sería totalmente contraproducente (e inaceptable desde el punto de vista ético) ridiculizar *al alumno* por cometer un error con un puntero, por ejemplo.

En este artículo se incluye una versión ligeramente reducida de dicho documento. Hay que advertir que el lenguaje utilizado en el mismo, lógicamente, no responde al estilo habitual en un documento científico como el presente. No obstante, hemos decidido incluir los textos aquí sin modificaciones, puesto que alterarlos para darles un aspecto más académico desvirtuaría notablemente la utilidad de este artículo.

A continuación se incluye, pues, una selección del texto final; la versión completa está disponible en [2]. En un apartado final se ofrecerán algunas conclusiones e información sobre el efecto que hasta ahora ha tenido el documento entre el alumnado de la Escuela Universitaria de Ingeniería Técnica en Informática de Oviedo.

2 De la programación propiamente dicha

2.1 Ignora los mensajes de error

Los compiladores, los sistemas operativos, etc. emiten mensajes de error sólo para que los usen sus creadores, o para justificar sus sueldos.

Leer esos mensajes, además, lleva un tiempo precioso que se podría dedicar a escribir más código, por lo que atentan contra nuestra productividad.

Y por último, para entender esos mensajes hacen falta conocimientos informáticos -cosa que no debemos cultivar, ya que en realidad todos estamos estudiando para jefes-, y encima... ¡la mayoría de las veces esos mensajes están en inglés! No caigas en la trampa. Ignóralos.

2.2 Ignora las advertencias, "warnings" o "hints"

Al igual que en el caso de los errores, estos mensajes son difíciles de entender (por lo menos más difíciles que los mensajes cortos de los móviles), y encima suelen estar en inglés.

Es más, ignorar los "warnings" le da a uno una pátina de programador profesional que no tiene miedo de los ordenadores. ¿Qué mejor demostración de madurez como programador que presentar un programa que al compilar da decenas, no, cientos de warnings, sin que su autor se inmute? Se nota que es un programador experimentado, que no se amilana por nada, y que además está demasiado ocupado para perder el tiempo con tonterías.

2.3 Escribe el código directamente sin pensar

Al final, no nos engañemos. ¿Qué es lo que estamos construyendo? Un programa. ¿Qué es lo único imprescindible en un programa? El código. ¿Qué es lo que de verdad funciona? El código. No hay que perder ni un minuto en usar medios arcaicos como lápices, bolígrafos o papel. Tú eres un miembro de pleno derecho de la generación WAP. No haces el ridículo escribiendo costosas sílabas, ¿verdad? Pues no hagas el ridículo pensando en nada, cuando hay tanto código por escribir.

Cuanto antes te quites de enmedio este rollo de programar, mejor. Y eso, muñeco, sólo lo puedes acelerar escribiendo más y más líneas.

2.4 Aunque el código no compile o no funcione, sigue escribiendo

Es sabido que los mensajes de error son una interrupción inadmisible, una traba estúpida a nuestro trabajo. ¿Qué puedes hacer si tienes un error de compilación? Ya hemos visto que leérselo y comprenderlo no es una opción válida.

Se puede intentar hacer algún cambio aleatorio en el código fuente, a ver si hay forma de engañar a ese estúpido compilador. Pero si eso no funciona, no pierdas más tiempo. NO, no caigas en la tentación de leer el mensaje de error o intentar comprenderlo (tú tienes un prestigio al que te debes). Sigue escribiendo código, que es de lo que se trata para acabar con esta asquerosa asignatura. Más adelante, ya arreglarás el error. Es sabido, además, que los errores tienden a desaparecer solos con el tiempo si no se los mira mucho. Cuando eso pase, ya tendrás el programa casi acabado. Compilarás, ejecutarás, y si probases (que tampoco hace falta) verías que todo funciona perfectamente.

Si el código compila, eso ya es el paraíso; nada nos impide seguir escribiendo código para liquidar de una vez esta odiosa práctica. El que el programa no haga lo que se quería que hiciera no es tan importante, ya lo arreglaremos más tarde, cuando esté acabado. Además, siempre puede pasar que en un golpe de suerte los profesores cambien el enunciado de la práctica y entonces sí que encaje con nuestro programa, así que no hay que arriesgarse a hacer trabajo inútil arreglando un programa que va descarriado.

2.5 Si el código tiene un error que no se produce siempre, ignóralo y sigue escribiendo

Mejor aún que en el punto anterior. Si el error no se produce siempre, va a ser difícil de encontrar, y además puede que en la demostración de la práctica no salga, y además puede que desaparezca solo si no lo miras. Así que no te preocupes, seguro que no es nada grave.

2.6 Si el código tiene un error que se produce siempre, cambia cosas aleatoriamente hasta que desaparezca

Ya hemos hablado en contra de pararse y pensar. Si hay un error que, por algún capricho, quieres eliminar, simplemente prueba a escribir el mismo código de otras formas diversas. Esto tiene la ventaja de que a lo mejor se soluciona, y además habrás conseguido que se solucione: 1) sin entender cuál era la causa, y 2) sin dejar de escribir código. Claramente, este es el comportamiento más profesional.

2.7 Construye enormes porciones de código sin compilar / ejecutar / probar

No compiles con frecuencia; no des pasos pequeñitos. Tú eres un profesional, y tienes que dar pasos de gigante. Escribe miles de líneas de código, y ya después compila. Así será mucho más entretenido buscar los errores de compilación y arreglar el código, lo que constituye un excelente ejercicio.

Respecto a ejecutar el programa que escribes, si intentas tener siempre un programa que funcione parcialmente, descubrirás los errores muy pronto, y además al haber hecho pocas modificaciones desde la última vez, te será demasiado fácil saber dónde has introducido el nuevo error. Esto sólo lo hacen los miedicas. Un verdadero programador hace el programa entero, y luego lo digiere entero, como una boa. Nada sustituye a la maravillosa sensación de buscar un error que se oculta en las últimas 10.000 líneas de código que has escrito; si sólo son 10 ó 20, la cosa no tiene ciencia.

2.8 No escribas comentarios, salvo los obligatorios

Ya lo hemos dicho antes. ¿Cuál es el objetivo de todo esto? Hacer un programa. ¿Y de qué consta un programa? De código. Todo código que no sea ejecutable no es realmente necesario. Poner comentarios explicando algo es un insulto a la inteligencia de un programador; cualquiera que vea un programa, si conoce el lenguaje de programación, sabe perfectamente lo que hace ese programa, cómo y por qué.

Si hay comentarios obligatorios (descripciones de funciones y toda esa morralla), esos sí hay que ponerlos, aunque no se tenga nada interesante que decir. A los profesores les gustan esas tonterías y te pondrán más nota.

2.9 Ignora los enunciados

Los enunciados y las especificaciones son un rollo. Hablan de problemas que no nos interesan, son largos, prolijos, y en realidad no son más que un simple ejercicio de narcisismo de los profesores para que veamos cuánto creen que saben. Limítate a echar un vistazo, decide más o menos qué te están pidiendo, y es suficiente; seguir leyendo el enunciado, o volver a leerlo más adelante, es un obstáculo en nuestra verdadera misión. Que no es otra, por supuesto, que escribir el código. Por tanto, una vez intuido más o menos lo que se pide, entierra el enunciado en el fondo de la pila de papeles más alta que tengas, o en las profundidades del directorio temporal de tu ordenador.

2.10 Ignora las normas de programación y presentación

Las normas que dicen cómo debe escribirse el código, o cómo debe presentarse la práctica, son un ejercicio de prepotencia por parte de los profesores. A ellos les gusta dominarnos, obligarnos a hacer cosas que no tienen ningún sentido, y por eso escriben normas. No te prestes a ese juego. Todas esas normas son totalmente innecesarias, ya que aun sin ellas las prácticas entregadas cumplirán los requisitos de calidad exigibles. Respecto a la facilidad de manejo por su parte, ¿no cobran para corregirlas? No te molestes siquiera en poner tu nombre o tu curso en las prácticas, ya que en realidad no tienen mayor dificultad en recordar tu cara, tu estilo inconfundible de programación, y sabrán que la práctica es tuya de todas formas.

2.11 Escribe la documentación al final

¿Cómo vas a escribir un documento describiendo un programa que no existe? Por otra parte, ¿qué sentido tiene escribir documentos para ti mismo sobre lo que vas haciendo? ¡Todo lo que puedas leer en ellos, ya lo sabes, puesto que lo habrás escrito tú! Así que el único motivo para escribir documentación de un programa es que los profesores la piden. Y eso puede solucionarse el día anterior a la entrega. Si, además, haces el programa durante los días previos a la fecha de entrega, no tendrás el problema de que se te olvide algo y necesites mirar documentos; todo estará muy reciente en tu cabeza.

2.12 No aprendas a utilizar el depurador ni otras herramientas

Si se produce un error en tu código, al fin y al cabo todo lo que sabes sobre programación te lo han enseñado los profesores. ¿De quién es, pues, la culpa de que tú tengas un error? Es del profesor; por tanto, que sea él quien lo busque y lo solucione. Los errores de programación son algo excepcional, cuando seas profesional no tendrás que enfrentarte a ellos, por lo que en tu etapa de formación es normal que no emplees tiempo ni esfuerzo en aprender a arreglarlos.

3 De la relación con el profesor

3.1 No pidas ayuda

Si no sabes hacer algo, si tienes alguna duda, si te has perdido, nunca pidas ayuda, nunca hagas preguntas en clase, ni vayas a ninguna tutoría. Hay miles de razones para esto, pero aquí daremos sólo algunas:

- Si vas a una tutoría a hacer preguntas, estás admitiendo que eres estúpido.
- Si vas a una tutoría a hacer preguntas, les darás trabajo y te cogerán manía.
- Si no entiendes algo, la culpa es del profesor. Pero ¿y si vas y te lo explican y sigues sin entenderlo? Entonces, la culpa pasará a ser tuya, ¿podrás soportar esa vergüenza?
- Si preguntas, puede quedar en evidencia que no sabes algo que deberías saber. Es mejor continuar en la ignorancia que correr ese riesgo.
- Si preguntas en clase, no sólo es que te oiga el profesor. Es que, claro, también te oirán tus compañeros. Y es evidente que tú vas a ser la única persona en la sala que necesite esa explicación, porque los demás, que nacieron aprendidos, seguro que lo entienden. Con lo cual seguro que pasan a opinar que eres estúpido.
- Si preguntas en clase, a lo mejor el profesor te resuelve la duda. Pero claro, obtendrá una valiosa información; a partir de las preguntas de los alumnos, puede que se dé cuenta de que su explicación no ha sido buena, de que la organización del curso no es adecuada, de que está yendo demasiado rápido, o cualquier otra cosa. ¿Acaso vas a darle esta información gratis? ¿Y si se le ocurre utilizar esa información para mejorar sus explicaciones? No seas insensato, no le facilites las cosas. ¿Acaso te beneficia en algo que vaya convirtiéndose en mejor profesor?.

Conclusión: nunca pidas ayuda ni acudas a una tutoría. Es mucho mejor confiar en tus compañeros, ya que al fin y al cabo están en tu situación, mientras que el profesor es tu contrincante, tu oponente, tu enemigo.

Sólo hay dos excepciones a esta regla: puedes preguntar si te has encontrado con algún problema

y no te da la gana buscarlo. Entonces, sí está justificado que acudas al profesor para que sea él el que haga tu trabajo. Si se niega, critícale en los pasillos.

La segunda es que se puede acudir a tutorías en la semana previa a la entrega del programa. Estará lleno de gente, el profesor se dedicará en exclusiva a atender alumnos obviando otras tareas, y además si no te atiende podrás criticarlo en los pasillos. Cada vez que vayas, tendrás que esperar durante horas, y eso te permitirá relacionarte con compañeros en la cola. Todo son ventajas.

3.2 Nunca describas un problema en detalle

Si a pesar de todo decides saltarte el punto anterior, y pides ayuda, ten en cuenta una regla de oro que te será también muy útil en tu vida profesional (no en vano la siguen multitud de profesionales y usuarios de la informática). NUNCA describas un problema en detalle.

Ejemplo. Si durante el desarrollo de un programa se produce algún suceso que te resulta desagradable, acude al profesor y dile: "Ayer me pasó algo que no me agrada". El pondrá cara de esperar más datos; aguanta, no digas nada más. No se te ocurra entrar en detalles como estos:

- Si el suceso desagradable se produjo durante la compilación del programa o durante la ejecución.
- Si el suceso desagradable hizo que el programa terminase súbitamente su ejecución o bien hizo que la ejecución continuase indefinidamente, o simplemente el programa no hizo lo que esperabas que hicera desde un punto de vista funcional.

Otro ejemplo. Si el suceso desagradable se produjo durante la compilación, no le digas al profesor el mensaje de error y la línea en la que se produjo dicho error. Dile sólo algo como "Me daba no sé qué error, o algo".

Otro ejemplo. Si el suceso desagradable se produjo durante la ejecución, y provocó la súbita terminación del programa, nunca anotes el mensaje producido, ni le digas al profesor el texto del mismo. Di simplemente "Me daba no sé qué error, o algo".

Evidentemente, si el suceso desagradable consistía en que el programa no hacía lo que esperabas, no se te ocurra decirle al profesor en qué situación exacta lo producía (si era al leer un fichero vacío, o antes o después de que pasase alguna otra cosa). Evita una descripción como "El error se produce siempre que cargo un segundo fichero y el primero estaba vacío". Tú di sólo la frase mágica: "Me daba no sé qué error, o algo".

¿Lo vas pillando?

3.3 Lleva siempre los fuentes equivocados

Supongamos que, contra todos nuestros consejos anteriores, acudes al profesor, y además vas a preguntarle por un problema concreto. Si el profesor se niega a examinar tus fuentes, podrás criticarle en los pasillos y advertir a futuros alumnos contra él, pero puede darse el caso de que el tío te reciba y tú acabes entrando y preguntando. Haces mal, pero la situación aún tiene arreglo: procura llevar los fuentes equivocados. Por ejemplo, si tienes un problema, y haces diversas modificaciones que no dan resultado, o incluso generan problemas nuevos, acude con los últimos fuentes, pero pregunta por el problema original. Así, el profesor buscará inútilmente un error cuando se le producirá otro. Entonces di algo como "Ah, bueno es que después probé una cosa. Quita esa línea de ahí..."

Si manejas este proceso con habilidad, puedes realizar toda una sesión de codificación allí mismo, en las tutorías. Esto es particularmente aconsejable si hay otros compañeros esperando fuera. El profesor cogerá mala fama, trabajará más tiempo, harás que el resto de los alumnos tengan más trabas para avanzar, con lo que el nivel de la clase se mantendrá en cierto equilibrio y así no pensarán en ampliar el temario en futuros años... Tómatelo como un servicio a la comunidad.

3.4 No aísles el problema

Volvamos a suponer que eres un irresponsable y acudes a una tutoría. No se te ocurra aislar el problema antes de ir. Si se produce un error en un fichero de entrada de 1 MB, no intentes ir probando con ficheros más pequeños hasta acotar qué produce el error, ni intentes crear un mini-

programa que reproduzca el mismo error. Eso podría permitir al profesor averiguar el problema de manera certera y rápida. Es mucho mejor que se lea miles de líneas de código y que haga trazas con centenares de miles de pasos. De este modo, ejercitará notablemente sus artes adivinatorias y podrás verificar su capacidad de deducción. Por supuesto, si se niega a buscar tu error, critícale en los pasillos.

3.5 Usa el correo electrónico con habilidad

Hay preguntas que son prácticamente imposibles de resolver por correo electrónico, si se hacen bien. Cultiva este arte. Haz que tu pregunta sea totalmente inconcreta. Ejemplo: "Me da no sé qué error, o algo. Aquí te mando el fuente". También puedes hacer una pregunta un tanto más concreta, pero no mandar el fuente: "En mi clase TDispositivo en el constructor me da no sé qué error, o algo". Si el profesor no resuelve el problema, critícalo en los pasillos por incompetente.

Por supuesto, escribe el mensaje de corrido y mándalo; no lo leas por segunda vez intentando ponerte en el lugar del profesor. Eso te podría llevar a detectar errores, y no es eso lo que se pretende.

3.6 Eskríbelo todo cn abrvtrs o konsonantes ekstrañas

Los profesores son unos puretas. Tú eres generación WAP. O si no, al menos serás un poko radikal. Intenta escribir mensajes que no resulten fáciles de leer. El tío, aunque crea que no, tendrá que hacer un esfuerzo extra, y además en la pantalla de un ordenador, ante la que quizás lleve varias horas. Eso facilitará su concentración y aumentará su buen humor.

3.7 Comete faltas de ortografía

La ortografía es una burda convención; hasta Juan Ramón Jiménez ponía jotas donde le daba la gana, y hasta Gabriel García Márquez abogaba por eliminar la ortografía. Evidentemente, tú no eres menos que ellos, así que tienes el mismo derecho a escribir como quieras.

¿Que no tienes costumbre de cometer faltas de ortografía? Es bien fácil. Puedes empezar por el ejercicio más fácil, más frecuente y más provechoso: "Haber si me sale". Ya, ya sé; la forma correcta es "A ver si me sale" (como diciendo "veamos si me sale"), pero ¿no encierra una brutal poesía esa brusca contracción de dos palabras en una, esa alteración semántica de utilizar el verbo "haber" sin significado alguno, esa hermosa palabra que arroja simultáneamente una B y una H (las letras más peligrosas de la ortografía española) a los ojos del lector? Si te apetece darle al profesor una bofetada y no tienes arrestos, escríbele un "Haber si puedo ir mañana a tutorías", que es lo más parecido.

3.8 No te identifiques

El correo electrónico tiene otra cosa divertida. Puedes empezar a largar sin que el tío sepa ni de qué grupo eres, ni quién eres. Todo se arreglará si en ese caso lo tuteas, porque así aumenta la familiaridad y no hace falta el nombre. Mejor aún es escribir un mensaje en el que no sepa ni de qué titulación eres. Sí, por ejemplo: "¿Cuál es la fecha de entrega para la primera práctica? Firmado: Fefu". Si el profesor da más de una asignatura en diferentes carreras (cosa extraordinariamente probable), será un buen ejercicio para él coger las listas de alumnos de todas ellas y buscar algo que pueda casar con "Fefu".

4 Y, sobre todo...

4.1 Deja el trabajo para el final

Desde el primer día, el profesor insistirá en intentar pedir cosas para la semana siguiente. Intentará avisar de que hay que ir haciendo el trabajo a un ritmo constante y decidido desde el principio.

No le escuches.

La programación de ordenadores, aunque es una disciplina joven, tiene ya sagradas tradiciones, y una de ellas es la prisa en los días previos a una entrega. Evitar ese estrés sería una nefasta preparación para tu vida profesional. Deja que el

trabajo se acumule, desoye las advertencias o los signos de retraso. No interrumpas tu vida ni dejes de ir a esquiar para recuperar tiempo perdido. Y cuando ya estés al límite del desastre, cuando falten dos semanas para la entrega de un programa estimado para cuatro meses... entonces empieza a escribir código como si no hubiera un mañana.

¿Qué aliciente tendría esta profesión si los programas se construyeran sin prisa y sin pausa? ¿Qué sería de la imagen del melenudo barbudo (o melenuda barbuda) maloliente (por no tener tiempo para afeitarse / cortarse las puntas / ducharse) con una camiseta de Megadeth (en las camisetas jeviatas se ven menos las manchas, gracias a su color negro y a los complejos dibujos), que lleva 48 horas ininterrumpidas dándole a la tecla? ¿Estarías acaso preparado para ir a la Campus Party y matar monstruos en una pantalla, con el culo sobre una silla de melamina y la cabeza bajo un techo de lona a 35ºC, durante tres días seguidos? ¿Cómo podríamos sentirnos un poquito héroes si todos los días nos vamos a dormir cuando nos entra el sueño? ¿Qué sería de las fábricas de Cocacola, qué sería de Juan Valdés, qué sería de las refinerías de cafeína que destinan la mitad de su producción a los programadores? ¿Acaso Sandra Bullock o Robert Redford cuando hacían de hackers se sentaban con sus apuntes al lado del ordenador, pensaban, tecleaban sin prisa y a la hora habitual se iban al gimnasio o al bar de la esquina, y así un día tras otro durante cuatro meses? ¿Acaso el tío ese de "Operación Swordfish" habría roto la clave del Pentágono si no hubiera estado un esbirro del Travolta apuntántole con una pistola mientras otra esbirra del Travolta lo distraía?

No, amigo, no. Para vivir tranquilo, te habrías matriculado en otra carrera. ¿Te imaginas ir a clase y que cuando el tío diga "Quién tiene hecho nosequé" tú puedas decirle "yo", y que cuando explique lo siguiente que hay que implementar tú estés atendiendo y entendiendo lo que dice porque lo anterior ya lo tienes funcionando?

Eso es para empollones y flojos de espíritu. Ya lo sabes. Deja todo el trabajo para el final.

4.2 Copia las prácticas

Copia las prácticas; ya sean las actuales por un compañero de clase, o prácticas del año pasado. Cada profesor puede tener que corregir varias decenas de prácticas; es difícil reconocer similitudes entre tantos programas. Si las reconocen, además, no es fácil demostrarlo, recurre y lleva el caso hasta el Tribunal Constitucional. Te llevará muchísimo más tiempo y esfuerzo que hacer las prácticas, pero el caso es demostrar que eres más listo que el profesor y no dar nunca el brazo a torcer.

La sensación de ahorro de trabajo que da copiar una práctica (o parte de ella) es inigualable y merece la pena. No importa que luego quedes en evidencia en la demostración, o en el examen práctico, o que en el fondo no hayas aprendido lo necesario, ya que el objetivo de las prácticas no es el aprendizaje, sino sólo la obtención de unos asquerosos créditos. Por tanto, copia lo que puedas. Si un profesor te suspende por copiar, critícale en los pasillos.

5 Efectos del documento en la comunidad

La dirección de la página web que contiene los “anti-consejos” fue comunicada, vía correo electrónico (un viernes), a los alumnos de 3 de los 14 grupos de prácticas de la asignatura Estructuras de Datos y de la Información, después de que presentasen el primer módulo de prácticas de esta asignatura. En el número del lunes siguiente del boletín informativo electrónico editado por los alumnos, InfoEUITIO [3], que cuenta con unos 300 suscriptores, apareció una reseña del documento; y varios profesores de otras asignaturas han sugerido su lectura. Desgraciadamente, esta acogida no estaba prevista y no hay estadísticas de acceso a la página, pero cabe suponer que el documento ha alcanzado un notable grado de difusión. El autor ha recibido varios mensajes al respecto, tanto de alumnos como de profesores.

En especial, resulta curioso apreciar que la respuesta de los alumnos es muy positiva; se leen el documento, les gusta, no se sienten ofendidos

en absoluto, admiten sin paliativos que se ven retratados en gran medida, y han contribuido mucho a su difusión. Un alumno sugería que, dado que InfoEUITIO no llega a todos los alumnos (sólo a los que se suscriben), debería enviarse el enlace a las listas de correo de las diversas asignaturas (Metodología de la Programación I y II, Estructuras de Datos y de la Información, etc.)[2]

Es pronto para saber si realmente estas instrucciones tendrán algún impacto en la calidad del trabajo que desarrollan los alumnos. Pero sí podemos concluir que este estilo de presentación de la información ha permitido alcanzar un sorprendente grado de penetración y una respuesta extremadamente positiva.

Referencias

[1] Agustín Cernuda del Río. *Informática - Microsoft Word (II). Cómo NO utilizar un procesador de textos*. Noviembre de 2001.

[2] Agustín Cernuda del Río. *Cómo NO realizar una práctica de programación*. http://www.agustincernuda.com/noprog.html

[3] Delegación de alumnos de EUITIO. http://petra.euitio.uniovi.es/~delegaci

[2] De hecho, su difusión se ha extendido a entornos profesionales externos a la Escuela, y es difícil conocer en este momento hasta dónde alcanza.