Hacia la Evaluación Continua Automática de Prácticas de Programación

Juan Carlos Rodríguez del Pino, Margarita Díaz Roca, Zenón Hernández Figueroa, José Daniel González Domínguez
Dpto. de Informática y Sistemas
Universidad de Las Palmas de Gran Canaria
{jcrodriguez,mdiaz,zhernandez,dgonzalez}@dis.ulpgc.es

Resumen

En el contexto de implantación y aplicación del Espacio Europeo de Educación Superior se hace necesario disponer de herramientas docentes que faciliten el seguimiento y la orientación personalizada y continuada del proceso de aprendizaje del alumno. En este trabajo se presenta un sistema de evaluación continua automática de prácticas de programación, vía web, basado en la realimentación. De esta manera, la evaluación proporciona información inmediata al alumno de los errores cometidos en las prácticas, permitiendo que puedan ser corregidas y reevaluadas repetidas veces. Este esquema establece un marco de evaluación, donde el alumno puede aumentar su motivación y decidir la nota que desea obtener. Los errores susceptibles de ser identificados automáticamente son de diversos tipos: incumplimiento de especificaciones de interfaz, errores de estilo y de diseño de codificación, errores de funcionalidad, errores graves de ejecución e insuficiencias en las pruebas.

1. Introducción

En las titulaciones de informática, y también en otras de perfil tecnológico, existen asignaturas que incorporan en su metodología la realización de prácticas en un lenguaje de programación. Normalmente, la elaboración de estos trabajos está supervisada por los profesores y se lleva a cabo en un laboratorio donde el alumno dispone del material informático necesario para su realización. Aunque, actualmente, la mayoría de ellos disponen de este material en sus casas, lo que les ofrece mayor flexibilidad, tanto espacial como temporal, a la hora de realizar las prácticas.

Los trabajos prácticos tienen unos plazos de entrega prefijados, tras los cuales son evaluados, pero hasta que se produce la revisión por parte del profesor, el alumno desconoce si es correcto o no. Dependiendo de las circunstancias de la asignatura —no es lo mismo una troncal de primer curso que una optativa de quinto—, la evaluación puede requerir bastante tiempo y esfuerzo del profesor debido a la cantidad de alumnos y el número y complejidad de los trabajos. Este modo de actuación satisface el aspecto evaluativo, pero no facilita que el alumno aprenda de sus errores, lo que hace que se pierda una parte importante del potencial formativo que conlleva la realización de una práctica.

La adaptación de los métodos de enseñanza tradicionales al Espacio Europeo de Educación Superior plantea nuevos retos al profesor que están condicionados por el esfuerzo necesario para alcanzarlos. La disponibilidad de una herramienta docente que facilite el seguimiento y la orientación personalizada y continuada del proceso de aprendizaje del alumno posibilita superar este reto con un coste asumible.

En este trabajo se presenta un sistema de evaluación continua y automática de prácticas de programación vía web basado en la realimentación. Esta herramienta lleva en funcionamiento desde el curso 2004/2005 en diversas asignaturas del primer y segundo curso de las titulaciones de informática de la Universidad de Las Palmas de Gran Canaria, abarcando a unos 650 alumnos por curso académico.

La motivación para desarrollar este sistema de evaluación/enseñanza ha sido que el profesor pueda realizar una evaluación continua y disponer de información personalizada sobre cada discente, además de facilitar la adquisición de conocimientos al alumno mediante una información detallada de los fallos cometidos. Con este sistema de evaluación se estimula al

además de facilitar la adquisición de conocimientos al alumno mediante una información detallada de los fallos cometidos. Con este sistema de evaluación se estimula al alumno a corregir sus errores y a comprobar de inmediato sus progresos.

1.1. Estado del arte

Varios grupos de profesores de diversas universidades españolas han empezado a incorporar en la evaluación de sus asignaturas algún tipo de herramienta automática [10]. En general, se centran en valorar los conocimientos teóricos adquiridos por los alumnos mediante pruebas tipo test, o bien en el desarrollo de una herramienta de evaluación automática de prácticas con el fin de ahorrar tiempo y esfuerzo de corrección al docente, olvidando las ventajas que aporta la tecnología [2] en cuanto a la posibilidad de comunicar resultados y, en caso de que existan fallos, la posibilidad de su corrección —a los pocos que abordan esta última opción les falta integrar la herramienta con alguna plataforma de campus virtual [3]—. A nivel internacional se han desarrollado herramientas de evaluación automática con realimentación, bien dirigidas a lenguajes de programación concretos como C++ [1], Scheme [13], Ada [7], o bien dirigidas al manejo de estructuras de datos mediante ejercicios de simulación [9]. La que más se parece a la propuesta en este trabajo es BOSS [8], siendo la principal diferencia que la información que se muestra al alumno como realimentación es significativamente menor que la que dispone el profesor para la evaluación. La metodología empleada por los autores de BOSS hace más hincapié en proporcionar a los alumnos pruebas *offline* para comprobar sus prácticas.

2. Criterios típicos de evaluación de prácticas de programación.

Antes de abordar la evaluación automática de prácticas de programación, es necesario repasar brevemente los criterios típicos que se suelen usar en dicha evaluación, a fin de estar en condiciones de discernir el alcance potencial de la evaluación automática, identificando lo que es posible evaluar y descartando lo que no lo es, al menos por el momento.

La evaluación de prácticas de programación suele estar basada [6] en varios criterios: diseño, ejecución, cumplimiento de especificaciones, estilo y creatividad.

El criterio de diseño se refiere a aspectos tales como: organización, descomposición modular, selección de estructuras y algoritmos, etc. El criterio de ejecución se ocupa de los fallos en el funcionamiento del programa, tanto los que reflejan incumplimientos de las especificaciones, como los que se deben a un mal uso de los mecanismos del lenguaje. El criterio de cumplimiento de especificaciones tiene en cuenta las especificaciones no funcionales, como la obligatoriedad de usar, o no, determinadas estructuras o construcciones —por ejemplo, que un determinado subprograma tenga que ser recursivo. El criterio de estilo trata de todo aquello que ayuda a la legibilidad y comprensión de un programa —comentarios, sangrado, espacios, nombres,... Por último, el criterio de creatividad premia la implementación de soluciones especialmente interesantes o innovadoras.

3. Herramienta empleada

Como soporte para la evaluación automática se ha empleado el programa de Gestión Automática de Prácticas GAP [11][12] . Una de las características del GAP 2.0 permite la ejecución de herramientas externas de forma segura, mediante la simple configuración de líneas de comando del sistema operativo, lo que resulta muy útil a la hora de realizar las pruebas automáticas de evaluación.

Para la ejecución de las pruebas automáticas de un ejercicio basta con escribir un guión usando el lenguaje de *script* del sistema operativo; este guión se ejecutará en el momento de la entrega —si así se ha configurado—, llevando a cabo las pruebas que en el mismo se hayan dispuesto y generando información sobre los resultados, que el GAP se encargará de mostrar al alumno convenientemente filtrada y formateada. El profesor puede ejecutar el mismo guión, u otro distinto sobre todas las prácticas entregadas si desea obtener información distinta o actualizar la que recibió el alumno en el momento de la entrega.

Dado que el código ejecutado podría contener partes maliciosas, el programa GAP ejecuta estas

pruebas en un entorno con privilegios restringidos mediante una cárcel *chroot* [4]. En una ejecución, el sistema realiza los siguientes pasos:

1. Crea un directorio temporal en la cárcel.
2. Transfiere a ese directorio una copia de los ficheros entregados por el alumno y de los preparados por el profesor para las pruebas.
3. Prepara el *script* elaborado por el profesor para ser ejecutado en dicho directorio.
4. Transfiere el control al *script*, aunque encarcelándolo y controlando su ejecución.
5. Finalizada o detenida la ejecución se recogen los resultados y se almacenan.
6. Se elimina el directorio temporal.

Este trabajo, usando estas características del GAP, describe cómo se puede establecer un sistema de evaluación automática con programas específicos y la ayuda de herramientas externas.

4. La revisión automática

El primer aspecto a tener en cuenta a la hora de preparar prácticas que van a ser probadas automáticamente es la ausencia de ambigüedad en su especificación. Las tareas encomendadas deben ser claras y concretas. Una evaluación automática sólo es posible si la variabilidad permitida está bajo control. Se puede argumentar en contra que esto disminuye las posibilidades de creatividad, pero también es cierto que, especialmente cuando el alumno es novel en la materia, puede ser muy beneficioso encarrilarle hacia un conjunto más o menos limitado de opciones. La alternativa sería dejarlo frente a una multiplicidad de posibilidades que, si no es capaz de controlar, fácilmente puede conducirlo al desánimo y al abandono.

Algunas de las revisiones que se realizan requieren la utilización de un analizador sintáctico que proporcione información acerca de múltiples aspectos del código, relacionados fundamentalmente con el cumplimiento de las especificaciones y el estilo. Aunque puedan existir herramientas estándar, el desarrollo de un analizador específico integrado en las pruebas, que permita acceder programáticamente a la información que suministra, presenta muchas ventajas. En la experiencia que aquí se expone se han desarrollado analizadores simples para C, C++ y Ada.

La calificación automática se basa en la penalización de los errores cometidos. Esta evaluación evita castigar reiteradas veces por el mismo tipo de error, así como que la acumulación de errores en una parte de la práctica invalide la valoración de otras partes.

4.1. Incumplimiento de especificaciones

Esta tarea se realiza normalmente compilando el código suministrado por el alumno junto con el correspondiente código de prueba. Se comprueban aspectos como que los nombres de los procedimientos, funciones, paquetes, y otros módulos coincidan con los indicados o que los tipos o el orden de los parámetros son los establecidos en la especificación, pudiéndose alcanzar un mayor grado de detalle con el apoyo del analizador sintáctico —por ejemplo, se puede informar al alumno de que se implementa la función *X* cuando se pide la función *Y*.

Otros aspectos a comprobar son: si un subprograma es recursivo cuando se ha especificado que lo sea, si se usan funciones o procedimientos que no están permitidos —por ejemplo, se pide que se implemente un procedimiento de manipulación de ristras sin usar las funciones de la biblioteca, y se usan, o si se emplean funciones de entrada y salida cuando se pide que no se empleen. Esta tarea se lleva a cabo en base a la información suministrada por el analizador sintáctico y la captura y análisis de la salida estándar.

4.2. Análisis de estilo

El analizador sintáctico se encarga de comprobar distintos elementos de estilo del código entregado por el alumno, tales como:

- Número, extensión y distribución de los comentarios.
- Calidad del sangrado.
- Longitud de las líneas de código.

En muchos casos, existen herramientas que permiten la realización de pruebas que usan guías de estilo estandarizadas sin necesitar un analizador específico —por ejemplo, el compilador *gnat* de Ada tiene opción de hacer comprobación de estilo—, aunque se complica la integración con el esquema de revisión al tener que procesar la salida de dicha herramienta.

4.3. Análisis de diseño

El análisis sintáctico puede suministrar información relativa al diseño de la solución, tal como:

- Número de subprogramas definidos.
- Tamaño de los subprogramas.
- Definición de subprogramas que no se invocan.
- Uso de variables globales.
- Complejidad ciclomática.
- Empleo de estructuras de control inadecuadas, como *goto*.

Los compiladores actuales también pueden proporcionar información adicional, por ejemplo: si se declaran variables que no se utilizan o si existe código inalcanzable.

4.4. Pruebas de funcionamiento

Las pruebas se realizan como pruebas unitarias, estableciendo casos de prueba para cada función o procedimiento que debe implementar el alumno de forma que cada caso sea independiente del resto.

Estas pruebas proporcionan información sobre errores de incumplimiento de la especificación funcional, además de manifestar errores graves de ejecución. Cuando se encuentra un error del primer tipo, se informa al alumno del caso probado que ha generado el error para que pueda depurarlo y corregirlo. En los errores graves, también se informa al alumno de las circunstancias que lo han provocado, pero, a diferencia de los otros, no se puede continuar la ejecución del caso de prueba.

Algunos de los errores graves que se pueden producir son: punteros sin inicializar, división por cero, índice fuera de rango, bucles infinitos, acceso a ficheros no abiertos, etc.

La actuación ante los errores graves depende bastante del lenguaje de programación. En el caso de lenguajes como Java, C# y Ada, los errores se manifiestan como excepciones que pueden ser controladas mediante los mecanismos suministrados al efecto por el propio lenguaje —las excepciones incorporan información sobre las circunstancias en que se han producido a la que se puede acceder mediante funciones específicas. En lenguajes como C o C++, el tipo de error se puede conocer capturando las señales del sistema operativo, pero a diferencia de los otros lenguajes, la continuación del resto de los casos de prueba se hace más compleja.

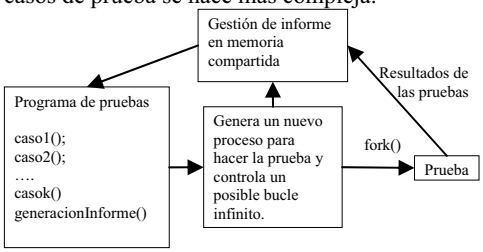

Figura 1. Gestión de errores graves en C y C++.

Para resolver este problema, cada caso de prueba en C y C++ se ejecuta en un proceso separado, lo que permite que un error grave en un caso no interrumpa la prueba del resto. El subprograma principal realiza la transferencia de información entre los distintos procesos usando memoria compartida y controla el tiempo de ejecución de cada caso para detenerlo si es necesario —por ejemplo, si hay un bucle infinito—, todo ello se ilustra en la figura 1. El subprograma en el que se ha producido el error se puede deducir de la función *backtrace* y de información sobre las funciones definidas por el alumno.

También, pueden producirse errores por mala utilización de los recursos —estos errores no siempre producen que la ejecución se detenga. Errores típicos de esta clase son, por ejemplo, en el lenguaje C: el mal uso de la memoria dinámica y el mal uso de ficheros.

La detección del mal uso de la memoria dinámica se hace sustituyendo las funciones de gestión de memoria (*malloc*, *free*, *realloc*,…) por funciones propias, que antes de realizar la tarea revisan su coherencia, pudiendo descubrir los siguientes errores:

- Toma de un bloque de memoria de tamaño erróneo (cero o exageradamente grande)
- Liberación de un bloque de memoria no tomado.
- No liberar bloques de memoria que ya no se usan.

Al igual que en el uso de la memoria dinámica, se pueden interceptar las funciones de gestión de ficheros (*fopen*, *fclose*, *fread*,…) sustituyéndolas por unas propias que comprueben la coherencia de su uso, pudiendo detectarse los siguientes errores:

- Usar un fichero no abierto.
- Abrir un fichero ya abierto.
- No cerrar los ficheros cuando termina la tarea a realizar sobre ellos.
- Uso incorrecto de parámetros.

4.5. Prueba de pruebas

Junto a la tarea de construir un código que cumpla unas especificaciones, a los alumnos se les solicita con frecuencia que añadan código que compruebe el correcto funcionamiento de la solución dada. La revisión de la prueba de los alumnos es muy compleja, ya que los casos a probar serían distintas implementaciones erróneas del código solicitado, para determinar si las pruebas encuentran el error. Normalmente, es más fácil realizar otras pruebas que obtengan información sobre la prueba del alumno. Una acción interesante es interceptar las llamadas realizadas desde las pruebas del alumno al código que prueba. Esto permite verificar cuántos casos se prueban para cada subprograma y revisar el tipo de casos examinados, aunque esto último es más complejo y difícil.

Con la experiencia, se ha comprobado que los alumnos tienden a escribir el código de prueba al final, después de haber comprobado —mediante el sistema de revisión automática— que el resto de su solución funciona bien, lo cual no es muy adecuado, por lo que se ha tomado como medida disuasoria el que, si no se realiza un número mínimo de pruebas, no se le informa de otros errores.

4.6. Herramientas externas

Existen herramientas muy útiles que pueden complementar las tareas de revisión descritas. En esta experiencia, para el caso concreto de C y C++, se han utilizado dos: *valgrind* [14] y *gcov*.[5]

valgrind. La gestión de memoria dinámica en C, la relación entre punteros y vectores, la simulación de parámetros de entrada y salida con punteros y demás flexibilidades del lenguaje hacen que sea fácil cometer errores al escribir el código. Por desgracia muchos de estos errores son espurios e irrepetibles, por ejemplo: cuando se usa una variable antes de darle un valor, puede que, por casualidad, el valor almacenado sea el que se espera como inicialización, con lo que el programa no muestra ningún síntoma; o cuando se accede fuera de rango en un vector, alcanzando una zona de memoria que ocupan otras variables del programa que no se están usando en ese momento, por lo que tampoco se manifiesta ningún error.

Este tipo de errores se da con más frecuencia de la esperada —la experiencia indica que un programa en C que ha pasado las pruebas de ejecución normales, con mucha frecuencia, contiene vicios ocultos que no se muestran.

valgrind es un intérprete de lenguaje máquina orientado a la detección de este tipo de fallos que, en la experiencia aquí descrita, se emplea como prueba final. Por desgracia, la salida que actualmente produce no es fácil de procesar automáticamente, aunque en un futuro cercano se espera que los informes sean en XML y puedan ser incorporados al informe general con facilidad.

gcov. Es un programa de GNU que se incluye con el compilador gcc y se emplea principalmente para realizar pruebas de cobertura. La prueba de cobertura permite conocer las líneas de código que no se ejecutan. En esta experiencia se emplea para saber si las pruebas del alumno ejercitan todo el código a comprobar. La salida de *gcov* es procesada para mostrar al alumno, en un formato adecuado, las líneas de código que sus pruebas no ejecutan.

4.7. Límites de la revisión automática

Aunque las pruebas enumeradas resultan bastante completas, a nadie se le escapa que existen determinados aspectos que son difícilmente evaluables de forma automática. Se trata fundamentalmente de aspectos cualitativos no cuantificables relacionados, por ejemplo, con cuestiones de estilo: si bien se puede medir el porcentaje y distribución de los comentarios, hoy por hoy resulta imposible garantizar que su contenido sea adecuado; otro ejemplo son los identificadores ¿cómo garantizar que el nombre de una variable es acorde a su función? Tampoco se puede garantizar al detalle que se estén usando los mecanismos más apropiados del lenguaje en cada situación. La valoración de la creatividad tampoco es un elemento fácilmente automatizable.

En este aspecto, pueden aparecer vicios derivados de las exigencias de las pruebas, como por ejemplo, que el alumno introduzca

comentarios sin sentido con el solo fin de garantizar el porcentaje mínimo requerido.

5. Cambio de los roles del profesor y el alumno por la revisión automática

La revisión automática ejerce un impacto visible en los papeles que juegan tanto el profesor como el alumno en el proceso evaluativo-formativo de los trabajos prácticos.

Sin la revisión automática, el profesor tiene dos momentos estelares en el proceso: la especificación del trabajo, a realizar al comienzo del proceso, y la revisión de las prácticas, entregadas al final; de estos dos, el segundo es extraordinariamente más costoso que el primero. Con la revisión automática, el momento inicial del proceso incluye no sólo la especificación, que, como ya se ha dicho, tiene que ser mucho más detallada, sino también el diseño y preparación de las pruebas; por el contrario, frente a este incremento de esfuerzo inicial, en la etapa final de revisión sólo tiene que ocuparse, en su caso, de los pocos aspectos que no son susceptibles de una evaluación automática. Esta revisión última sirve, no sólo para establecer una calificación definitiva —no debe ser costoso, ni divergir mucho de la propuesta automáticamente por el sistema—, sino también para mejorar las pruebas, en función de lo que se observe —fundamentalmente, cuando la calificación definitiva difiera significativamente de la propuesta. El profesor disminuye considerablemente el tiempo dedicado a la tarea repetitiva de evaluación pudiendo dedicar ese tiempo a establecer las pruebas automáticas.

En cuanto al alumno, su papel pasa a ser más activo. Sin revisión automática, hace su trabajo, lo entrega y se desentiende hasta que el profesor le comunica su calificación. Con revisión automática, el alumno recibe una valoración inmediata al entregar su ejercicio, como se muestra en la figura 2, y entra en un proceso de corrección/reentrega hasta que finalizan los plazos de entrega máximos establecidos.

Figura 2. Ejemplo de información mostrada al alumno en el momento de la entrega

6. Experiencia de uso

Como ya se ha mencionado, esta herramienta lleva en funcionamiento desde el curso 2004/2005 en diversas asignaturas del primer y segundo curso de las titulaciones de informática de la Universidad de Las Palmas de Gran Canaria, abarcando a unos 650 alumnos por curso académico. Como ejemplo, se pueden analizar los datos correspondientes a la asignatura "Metodología de la Programación II", de segundo curso del año académico 2006/2007. Para la segunda práctica, en la que participan 209 alumnos con 2.236 revisiones automáticas, la figura 3 muestra el número promedio de entregas que realizan, cada día, los alumnos que hacen alguna entrega ese día. Se observa que, como promedio, los alumnos, el día que están trabajando hacen 2.11 entregas. También se observa que el promedio de entregas se incrementa a medida que se acerca la fecha de finalización del plazo de realización de la práctica; ello es coherente con lo mostrado en la figura 4, que refleja que un porcentaje importante de alumnos tiende a entregar los últimos días, no aprovechando convenientemente la realimentación, aún haciendo más entregas promedio —entran en un ciclo de prueba y error.

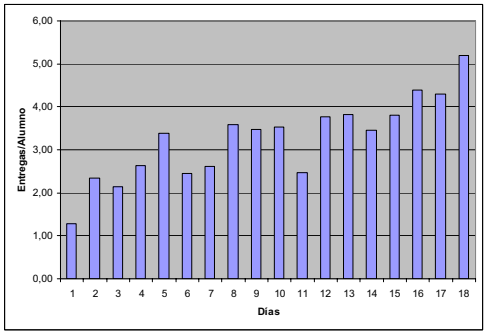

Figura 3. Entregas promedio por alumno que entrega ese día.

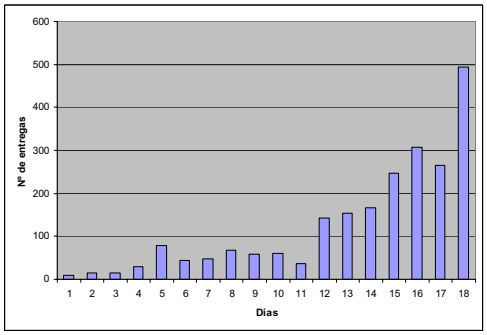

Figura 4. Número de entregas por día.

Por otro lado, la figura 5, refleja la incidencia de la realimentación en los alumnos que superaron la parte de la asignatura correspondiente al dominio del lenguaje en la convocatoria ordinaria de febrero de 2007 —y que no la habían superado por parciales. Es de destacar cómo el 73% de los aprobados usaron la realimentación proporcionada por el sistema y cómo esta realimentación ha sido más beneficiosa para los alumnos noveles que para los repetidores, lo que viene indicado por dos datos: que los alumnos noveles con realimentación son más de la mitad de los aprobados, y que la diferencia entre éstos y los alumnos noveles que aprobaron sin aprovechar la realimentación es muy superior a la que se da entre los repetidores, con y sin realimentación.

Figura 5. Incidencia de la realimentación en los alumnos aprobados en febrero de 2007.

7. Conclusión y perspectivas de futuro

La utilización de una herramienta de revisión automática de prácticas de programación es efectiva en la mejora de la calidad del esfuerzo del profesor al liberarlo de las tareas más tediosas y costosas de evaluación de código informático. Ello le permite dedicar más tiempo al diseño y mejora de las prácticas y sus pruebas. Además, se consigue una evaluación más homogénea y justa.

También es efectiva, sobre el rendimiento del alumno, al proporcionarle una realimentación inmediata de los resultados de su trabajo, que le permite mejorar su proceso formativo. No obstante, habría que conseguir que el alumno aproveche mejor esta herramienta docente a lo largo de todo el periodo de desarrollo de las prácticas, evitando el vicio generalizado de trabajar a última hora. En este sentido, se ha planteado premiar con mejor nota a los alumnos que entreguen con antelación a la finalización del plazo.

Con el tiempo, la herramienta se ha convertido, para los profesores que la usan, en un elemento importante y de difícil sustitución en el contexto de trabajo con grupos numerosos de alumnos.

Actualmente no se dispone de un seguimiento detallado del trabajo del alumno, que permita una mejor evaluación del coste de realización de las prácticas y una mayor autentificación en la autoría de éstas. Como mejora se tiene en perspectiva

incorporar herramientas que permitan hacer este seguimiento.

Referencias

[1] Ala-Mutka, K.; Uimonen, T. and Järvinen, H. *Suppporting Students in C++ Programming Courses with Automatic Program Style Assessment*. Journal of Information Technology Education. Vol. 3, pp 245-262. 2004.

[2] Barberá, E. *Aportaciones de la tecnología a la e-Evaluación*. RED. Revista de Educación a Distancia [en línea], Julio de 2006. en http://www.um.es/ead/red/M6 [consulta 18/01/2007].

[3] Durán, F.; Gutiérrez, F. y Pimentel, E. *El uso de herramientas de apoyo para la valoración de actividades prácticas de programación.* [en línea]. http://www.uca.es/... [consulta 19/02/2007].

[4] Friedl, S. *Go Directly to Jail*. Linux Magazin. December 2002.

[5] *Gcov* en GCC, the GNU Compiler Collection [en línea]. http://gcc.gnu.org/ [consulta 19/02/2007].

[6] Howatt, J. W. *On criteria for grading student programs*. ACM SIGCSE Bulletin, Vol. 26(3). September 1994.

[7] Jackson, D. and Usher, M. *Grading student programs using ASSYST*. ACM SIGCSE Bulletin, Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education SIGCSE '97, USA. Vol. 29(1), pp 335-339. March 1997.

[8] Joy, M.; Griffiths, N. and Boyatt, R. *The BOSS Online Submission and Assessment System* ACM Journal on Educational Resources in Computing (JERIC) Vol. 5(3), September 2005.

[9] Laakso, M. and Salakoski, T. and Korhonen, A. *The feasibility of automatic assessment and feedback*. Proceedings of Cognition and Exploratory Learning in Digital Age (CELDA 2005), pp 113-122. 2005.

[10] MOODLE. Moodle is a course management system (CMS). http://moodle.org/

[11] Rodríguez del Pino, Juan Carlos. *Gestión Automática de entrega de Prácticas vía web*. Actas de las VIII Jornadas de enseñanza Universitaria de la Informática. Cáceres, pp 53-58. Julio 2002.

[12] Rodríguez del Pino, J. C.; Hernández Figueroa, Z. ; González Domínguez, J. D. ; Díaz Roca, M. Experiencia de uso y características del programa Gestión Automática de Prácticas (GAP), Actas Conferencia IADIS Ibero-Americana, pp 35-42. Octubre 2005.

[13] Saikkonen, R.; Malmi, L. and Korhonen, A. *Fully Automatic Assessment of Programming Exercises*. ACM SIGCSE Bulletin, Proceedings of the 6th annual conference on Innovation and technology in computer science education ITiCSE '01. Vol. 33(3). June 2001.

[14] Valgrind [en línea]. http://www.valgrind.org/ [consulta 19/02/2007].