CUESTOR: Una nueva aproximación integral a la evaluación automática de prácticas de programación

Francisco P. Romero
E. U. de Ing. Tec. Industrial
Universidad de Castilla La Mancha
Avda. Carlos III, s/n
45071 Toledo
FranciscoP.Romero@uclm.es

Jesús Serrano-Guerrero
Escuela Superior de Informática
Universidad de Castilla La Mancha
Paseo de la Universidad, 4
13071 Ciudad Real
Jesus.Serrano@uclm.es

Hernán Pérez de Inestrosa
Escuela Superior de Informática
Universidad de Castilla La Mancha
Paseo de la Universidad, 4
13071 Ciudad Real
Hernan.PerezInestrosa@alu.uclm.es

Resumen

A pesar de que existen diversas aproximaciones para la evaluación automática de prácticas de programación, su aplicación fuera de los entornos en que fueron diseñados no siempre es posible. En este trabajo se presenta una nueva plataforma abierta que proporciona los mecanismos necesarios para realizar una evaluación completa de un ejercicio de programación realizado en C o en Java. Este proceso de evaluación incluye la verificación del cumplimiento de los requisitos especificados, el método de resolución, la calidad del código fuente y la comprobación del plagio. El funcionamiento de cada uno de los componentes de evaluación ha sido verificado de forma exhaustiva mediante la utilización de las entregas realizadas por los alumnos en años anteriores.

1. Introducción

Actualmente la enseñanza universitaria de los fundamentos de la programación de ordenadores está cada vez más basada en la metodología del aprendizaje por proyectos, es decir, el alumno se enfrenta cada vez más precozmente a la realización de programas en su entorno de desarrollo y a la verificación de su funcionamiento de forma casi inmediata. A consecuencia de esto, se produce una tendencia al abandono de la "programación en papel" para pasar a la "programación directa" en el ordenador. Esta forma de enseñanza ofrece un contacto directo y continuado a los alumnos sobre el entorno de programación así como la asimilación más sencilla de conceptos como la calidad en el estilo de codificación, fundamental en el trabajo de un programador.

De la misma manera que los alumnos evolucionan en su forma de hacer las cosas, también deben evolucionar los métodos de evaluación. El gran volumen de ejercicios resueltos por los alumnos semanalmente en un curso cuatrimestral de fundamentos de programación hace prácticamente imposible la corrección individual de cada uno de ellos, lo cual provoca que los alumnos no reciban de forma inmediata y directa la información necesaria sobre la evolución de su aprendizaje.

Para solucionar este problema se plantea la utilización de herramientas que permitan soportar algunos aspectos de la evaluación de un ejercicio. De esta manera se ayudará al profesor en su labor docente y al alumno en su evolución en la asignatura. El objetivo es complementar el estudio presencial mediante la disponibilidad de un recurso que puede ofrecer una información valiosa para el proceso de aprendizaje del alumno.

Existen diferentes aproximaciones sobre la evaluación automática de las prácticas de programación [10][11]. La cobertura de éstas suele ser parcial, es decir, sirven para evaluar algún aspecto individual, bien sea la calidad del código o bien el control del plagio en el caso de trabajos calificables. Existen herramientas integrales de evaluación de ejercicios [13], las cuales, o bien son desarrollos propietarios con una difícil adaptación a otros entornos, o bien cuentan únicamente con interfaces básicos de usabilidad reducida, no ofreciendo información sobre la evolución del alumno a lo largo del curso. Los entornos de aprendizaje virtual desarrollados en universidades que ofrecen enseñanza a distancia

ofrecen entornos mucho más atractivos pero más centrados en la enseñanza y en el soporte directo del aprendizaje que en la evaluación misma.

El recurso que se presenta en este trabajo es una nueva aproximación a la evaluación automática de prácticas de programación. En él se incluyen los métodos para realizar una evaluación integral de la práctica, es decir, codificación, evaluación del funcionamiento y del método de resolución, así como la detección del plagio entre prácticas entregadas.

El objetivo principal es lograr una evaluación lo más completa posible de los ejercicios de programación. Además, se persigue que la herramienta sea independiente del lenguaje y entorno utilizado para desarrollar el programa así como de la interfaz que se ofrezca al usuario. Es por ello que entre sus características cuenta con un diseño basado en componentes modulares que facilita una adaptación rápida a cualquier entorno de funcionamiento.

Con la utilización de esta herramienta se pretende conseguir por un lado disminuir la carga de evaluación de ejercicios que soporta el profesor de la asignatura y por otro lado que los alumnos reciban una información más directa tanto sobre la asimilación de los conceptos propuestos, como sobre su evolución durante el curso.

Este trabajo se estructura de la siguiente forma. En la Sección 2 se realiza una revisión de los recursos existentes para la realización total o parcial de la evaluación automática de un ejercicio de programación. En la Sección 3 se describen las características y funcionamiento del sistema planteado, tanto de forma global como de forma particular atendiendo a cada uno de los aspectos evaluables de un ejercicio de programación. En la Sección 4 se presentarán la aplicación práctica de la herramienta en proceso de enseñanza-aprendizaje, así como las verificaciones realizadas sobre los procedimientos de evaluación. Por último, la Sección 5 muestra las conclusiones y trabajos futuros.

2. Trabajos Relacionados

Entre la literatura científica existente se puede distinguir entre dos tipos de herramientas que están relacionadas con el recurso presentado. Por un lado, las que cubren parcialmente alguna de las funcionalidades propuestas y por otro, los sistemas que integran todas estas funcionalidades en una única herramienta. Problemas como la detección del plagio y la verificación de la calidad del código fuente han sido algunos de los que han suscitado más interés entre la comunidad científica durante los últimos años.

Las herramientas para la detección del plagio en prácticas de programación más conocidas son las que ofrecen una interfaz mediante servicios web como Jplag [10]. En ella ni el código fuente ni la propia aplicación está disponible si no es por el uso del servicio web ofrecido. JDup [9] es una aproximación desarrollada en España que utiliza un algoritmo de detección del plagio similar al de Jplag. Su principal virtud es la independencia que ofrece con respecto al lenguaje de programación utilizado. En este contexto, otros sistemas desarrollados en España son el sistema Heracles [6] y la plataforma Web presentada en [5].

En cuanto a la evaluación tipográfica del código, la base de todos los analizadores de estilo sobre código C se presentó en [4]. Más recientemente, en [2] se propone una herramienta que analiza el estilo de codificación de un programa en lenguaje C++. La herramienta, llamada Style++, puede ser utilizada tanto por los estudiantes como por el profesor con el fin de comprobar que reglas se cumplen y cuáles se incumplen en cuanto a calidad en la codificación.

Todas estas herramientas únicamente proporcionan una solución parcial a la evaluación automática de los ejercicios de programación. Es por ello, que a pesar de ofrecer una buena base de trabajo y ser referencia en sus diferentes ámbitos, para construir un sistema de evaluación integral de prácticas de programación es necesario ir más allá, incluyendo mecanismos de comprobación del funcionamiento e integrando en una sola herramienta todos los mecanismos de evaluación disponibles.

El principal sistema de evaluación completa de prácticas de programación es Ceilidh [13] y su evolución CourseMaster. Esta herramienta pretende ser una aproximación de evaluación automática accesible tanto para el estudiante como para el profesor. El sistema está dividido en cuatro módulos: evaluación tipográfica, evaluación dinámica del funcionamiento mediante la ejecución de test unitarios, evaluación de las características básicas de la solución y un módulo de detección de plagio. El principal inconveniente

de esta herramienta reside en su gran proyección comercial, es por ello, que su código fuente y sus especificaciones técnicas no están accesibles. Además el sistema es suficientemente inflexible con lo que la docencia se tiene que adecuar a la herramienta en vez de realizarse una adaptación en el sentido contrario. En resumen, la implantación, el mantenimiento y la ampliación de CourseMaster en otros entornos es muy costosa y muchas veces inviable. Por esta causa existen universidades como la Universidad Oberta de Catalunya que con su proyecto SICAP [8] afrontan la construcción de sistemas de funcionalidades similares a CourseMaster. En SICAP se aborda tanto la evaluación automática de programas como la enseñanza virtual de la programación. Otro trabajo de índole similar es el presentando en [12] en el que se propone un sistema de evaluación continua de prácticas de programación basado en GAP, un sistema de gestión automática de prácticas. Esta herra-mienta no proporciona un mecanismo offline de evaluación de prácticas que no deban tener respuesta directa al alumno.

Como conclusión se puede observar que los trabajos existentes bien son aproximaciones parciales a un aspecto de la evaluación o bien son proyectos monolíticos cuya implantación y adecuación al entorno de aprendizaje es complicada. Es por ello que en este trabajo planteamos un sistema que ofrezca la posibilidad de automatizar la evaluación completa de la práctica y además sea fácilmente adaptable a cualquier entorno de desarrollo, lenguaje de programación, distintas interfaces de gestión de las prácticas, etc.

3. Características de la herramienta

CUESTOR es una aplicación desarrollada en Java que proporciona mecanismos de evaluación automática de prácticas de programación cuyos resultados son útiles tanto para agilizar la labor evaluadora del profesor como para informar al alumno de su avance en su proceso de aprendizaje.

El entorno de funcionamiento de la herramienta corresponde a la docencia en asignaturas de fundamentos básicos de programación que están programadas en los primeros cursos de carreras universitarias como Ingeniería Informática e Ingeniería Industrial. Los ejercicios a procesar son por lo tanto programas sencillos en los que se resuelven problemas básicos de programación en C o en Java.

En CUESTOR existen dos modos de funcionamiento básico, el modo individual que se ocupa de realizar la evaluación de un ejercicio único entregado por el alumno a través de una herramienta de gestión de prácticas, y el modo conjunto, que permite al profesor la corrección masiva de una entrega de prácticas a partir de un directorio en el que estén todas las prácticas almacenadas.

Los principios de diseño de la herramienta han sido los siguientes:

- Independencia: La herramienta evaluadora soporta varios lenguajes de programación. Actualmente está preparado para evaluación de programas en C y Java, pero no es complicada su extensión para evaluar programas en otros lenguajes. Esta independencia también se extiende al entorno de desarrollo empleado en la docencia y la aplicación de gestión de las prácticas que se utilice.
- Configurabilidad: Todos los parámetros, desde localización de prácticas, ficheros de prueba, ponderación de las calificaciones, etc. son configurables.
- Disponibilidad del código fuente: El código fuente del componente evaluador estará disponible bajo licencia GPL y accesible desde algún repositorio público[1] de software.

El núcleo de CUESTOR (Figura 1) está conformado por los siguientes módulos:

- Módulo evaluador: Se ocupa de verificar la práctica entregada teniendo en cuenta parámetros como el estilo de codificación, el funcionamiento correcto y el método de resolución. La calificación final vendrá establecida por la agregación de estos tres parámetros, cuyo peso específico será definido en la configuración.
- Módulo de detección del plagio: Permite comprobar si se han producido copias en las entregas realizadas.
- Procesador de Entregas: Permite extraer todos los ejercicios de una entrega realizada en otras herramientas como por ejemplo MOODLE y

[1] http://sourceforge.net/projects/cuestor/

verificar si cumplen la normativa establecida de forma semi-automática.

- Generador de Calificaciones: Permite generar las calificaciones obtenidas en formato CSV o XML con el fin de que puedan ser publicadas o importadas en otras herramientas.

3.1. Evaluación del Estilo de Codificación

El estilo de codificación es un apartado importante dentro de la enseñanza de la programación. Es por ello necesario evaluar cómo se adecuan los códigos fuente generados a las convenciones de codificación de software que se establezcan al principio de curso.

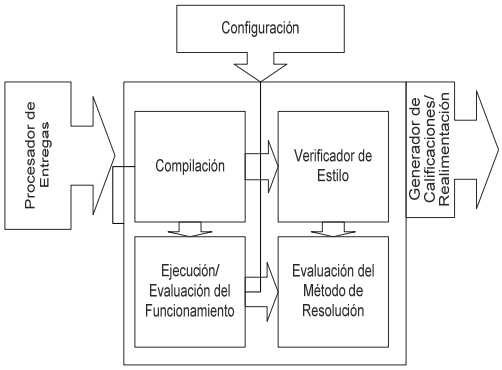

Figura 1. Componentes de CUESTOR.

En el caso de la programación en C se han establecido como parámetros evaluables la inexistencia de *"warnings"* asociados a la codificación, el nivel de comentarios [4] incorporando un mecanismo de comprobación de palabras con el fin de que no se puedan introducir comentarios sin un mínimo de sentido, el cumplimiento de las convenciones de organización del código y nombrado (similares tanto para C como para Java), así como otros parámetros que son específicos de cada lenguaje.

El método de evaluación del cumplimiento de las convenciones de código está basado en el número de cambios necesarios para obtener el fichero correctamente codificado a partir del fichero original entregado.

3.2. Evaluación del funcionamiento

La evaluación básica del funcionamiento del ejercicio consiste en la adecuación de las salidas de la práctica presentada a los requisitos especificados en su enunciado. En este caso existen tres alternativas para su evaluación, la evaluación interactiva, la evaluación mediante el análisis inteligente de la salida o la utilización de *frameworks* de pruebas unitarias. En los tres casos la ejecución del programa se realiza de forma automática, disponiendo cada práctica de un fichero con los datos de prueba necesarios para su ejecución. Estos datos corresponderán a las ejecuciones necesarias para comprobar el funcionamiento de la práctica. Esta ejecución automática cuenta con mecanismos de control que evitan la ejecución de códigos maliciosos y de sucesos como bucles infinitos (control del tiempo de ejecución del programa, ejecución dentro de un entorno controlado, etc.)

En el caso de la evaluación interactiva es el usuario/profesor el que califica el funciona-miento tras comprobar los resultados ofrecidos por la ejecución automática. Esto es especialmente conveniente para evaluar ejercicios en que existe cierta libertad en la generación de las salidas de los programas.

En el caso de la evaluación mediante el análisis automático de las salidas arrojadas por el programa se utilizan diversos algoritmos de reconocimiento de patrones en textos asumiendo error [3]. De esta manera se puede comprobar si las salidas del programa realizado por el alumno han sido las correctas, distinguiéndose entre un funcionamiento parcial o total del ejercicio según el número de casos de prueba que se ejecuten correctamente.

Por otro lado, en el caso de que el programa esté realizado mediante técnicas de Programación Orientada a Objetos también se hace posible la evaluación mediante la utilización de un *framework* de pruebas unitarias. Para ello se debe especificar la clase de Test o programa de Test que corresponderá a la comprobación del funcionamiento de la práctica. Los *frameworks* utilizados dentro de CUESTOR han sido *JUnit*[2] y *CppUnit*[3].

[2] http://www.junit.org/
[3] http://sourceforge.net/projects/cppunit/

3.3. Evaluación del método de resolución.

Este módulo es el más complicado en la herramienta y aún está en proceso de prueba. El objetivo es verificar si el programa entregado utiliza aquellos elementos que eran necesarios para resolver el problema de forma conveniente. Para realizar este módulo se han seguido y adaptado algunas de las propuestas realizadas en [13]. Actualmente se han implementado las diferentes comprobaciones.

- Comprobación de la corrección en los prototipos de las funciones (parámetros, tipos, parámetros por referencia), localización de la E/S, etc..
- Evaluación de los bucles utilizados en cuanto a número, funcionalidad y complejidad, con el fin de distinguir entre soluciones correctas e incorrectas y dentro de las correctas las que son eficientes de las que no lo son.
- Evaluación del número de componentes clave que se utilizan (estructuras de control, funciones, etc.) y su organización dentro del código, comparándola con la estructura básica de la solución o soluciones proporcionadas por el profesor.
- Utilización de resultados de programas externos de evaluación como *splint*[4] y *gcov*[5] que ofrecen resultados sobre los problemas estáticos que puedan existir en los programas como del código que no se ha ejecutado durante la realización de las pruebas, respectivamente. Un nivel bajo de código ejecutado en las pruebas indica que la práctica tiene excesivo "código sobrante".

Cada una de estas comprobaciones se pondera por configuración para determinar su aportación a la nota final.

3.4. Detección del Plagio.

Existen dos niveles de detección del plagio según sea la naturaleza del ejercicio a entregar. Si éste corresponde a uno de los ejercicios semanales la detección del plagio se realiza únicamente en un primer nivel. Si por el contrario la práctica es un trabajo más amplio se realizan las comprobaciones de primer y segundo nivel.

En el primer nivel de detección se utilizan únicamente técnicas de comparación de textos. Se comparan los ficheros fuente comprobando el número de cambios que se tienen que hacer para transformar una práctica en otra ignorando aquellos que pueden ser incluidos por el propio mecanismo de copia (espacios en blanco, nombres de las variables, etc.).

Las técnicas y componentes para la detección del plagio de segundo nivel se basan en las técnicas de comparación de estructuras que pueden verse especificadas en [10] y [9]. En primer lugar cada práctica entregada es preprocesada con el fin de obtener un código de referencia [7]. Esto se realiza mediante un procesamiento de comentarios, identificadores y mensajes que se transforman a una representación intermedia. A partir de estos resultados se realiza un análisis estructural del programa. Los resultados obtenidos de esta manera son mucho más detallados que los ofrecidos por las técnicas aplicadas en el primer nivel, lo cual permite reducir el número de falsos negativos que ofrecen la comparación de textos.

Tanto la evaluación del primer nivel como la del segundo nivel son dependientes del lenguaje. En esta última es necesario contar con una especificación en forma de gramática del lenguaje utilizado en el desarrollo de la práctica.

La modularidad del sistema permite incorporar un comprobador de plagio externo como *Plaggie* [1] cuyo código está disponible y su funcionamiento es bastante bueno sobre ejercicios escritos en Java.

4. Aplicación y Evaluación

La aplicación de la herramienta propuesta se ha realizado en dos entornos bien diferenciados. Por un lado se ha realizado su aplicación off-line, es decir, se han verificado los componentes evaluadores mediante la comparación de los resultados obtenidos en años anteriores en la corrección realizada de forma manual con los resultados que ofrece la evaluación automática. Por otro lado, la aplicación CUESTOR se ha integrado dentro de una herramienta en desarrollo de gestión de prácticas.

[4] http://lclint.cs.virginia.edu/
[5] http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

4.1. Verificación del funcionamiento

El funcionamiento de CUESTOR se ha verificado mediante la comparación entre las calificaciones otorgadas manualmente en los ejercicios de programación en C entregados en el curso 2008-2009 (asignatura de Fundamentos de Informática en Ingeniería Técnica Industrial) y las obtenidas automáticamente mediante el sistema. En los dos casos se utiliza una ponderación similar para los aspectos estilo, funcionamiento y método.

Las características de la evaluación han sido los siguientes. Se cuenta con una división de 6 módulos temáticos en la cual se propone una relación de 5 ejercicios que deben de ser entregados semanalmente. El número de alumnos matriculados en el curso es de 122. A estas entregas es necesario incorporar dos trabajos evaluables, dos exámenes parciales y 2 exámenes finales. El número total de ficheros evaluados es de 2930 ficheros distribuidos entre las diferentes entregas (Figura 2).

En cuanto a la verificación del estilo de programación los resultados han sido muy positivos. En primer lugar se han podido evaluar de forma automática el 100% de las entregas recibidas, y en segundo lugar el error acumulado ha resultado ser inferior 12%, lo cual puede ser achacado a la granularidad de la codificación.

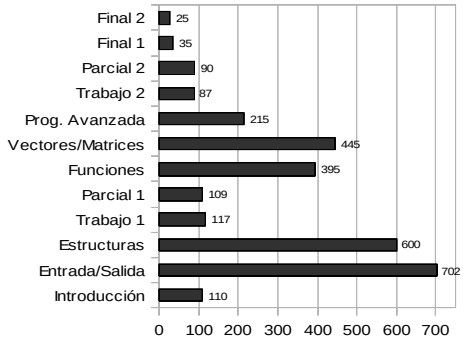

Figura 2. Nº de ficheros por módulo temático.

En cuanto a la verificación del funcionamiento se consigue un rendimiento adecuado en los ejercicios que han podido ser evaluados (cerca de un 80%). En este caso se consiguen valores cercanos al 100% en las evaluaciones realizadas de forma interactiva y cercanos al 85% en el caso de la realización automática de la evaluación. El 20% de los ejercicios que no han podido ser ejecutados se deben a la interpretación que dio el alumno a las indicaciones del enunciado.

Con respecto a la verificación del método de resolución los resultados no han sido satisfactorios dada la ambigüedad de algunos enunciados. Es por ello, que en este punto se plantea evaluación semi-automática dirigida por el profesor, además de la utilización de enunciados más completos y que incluyan ejemplos de ejecución.

La detección de plagio se ha realizado sobre los trabajos evaluables, es decir un total de 204 entregas divididas en dos trabajos distintos. Manualmente se habían detectado un total de 16 trabajos plagiados en un mayor o menor nivel. Utilizando la comprobación del plagio explicada anteriormente se han obtenido un 100% en la detección de esos plagios y además se han incorporado otros 5 trabajos que bien podrían ser considerados de esa manera tras una revisión más exhaustiva.

Los resultados obtenidos han sido satisfactorios en general (Figura 3), y muy satisfactorios si consideramos que una parte de la evaluación pueda ser asistida o comprobada por el profesor.

Dado que se ha comprobado la fiabilidad del sistema de evaluación, éste se ha utilizado de forma off-line por parte del profesor durante el primer cuatrimestre del curso actual. En los casos como exámenes y trabajos evaluables esta evaluación automática se ha realizado de forma interactiva.

4.2. Herramienta de Gestión de Prácticas de Programación.

Con el fin de poner en práctica la herramienta se ha integrado CUESTOR dentro de una herramienta web (desarrollada en Java) de gestión de prácticas de programación en desarrollo como es HUGIN. Esta herramienta es la que cumple las funciones de interfaz para los alumnos. En ella se pueden consultar los datos de la evaluación de cualquier entrega de forma inmediata y a su vez también pueden comprobar en cualquier momento la evolución del alumno en el aprendizaje de la asignatura. En la figura 4 se muestran los resultados ofrecidos por la herramienta en la evaluación automática de un ejercicio que consiste en determinar si un año es o no bisiesto.

HUGIN[6] está actualmente en fase de pruebas por lo que está siendo utilizado por una muestra de alumnos con características especiales (responsabilidades profesionales, últimas convocatorias, etc.).

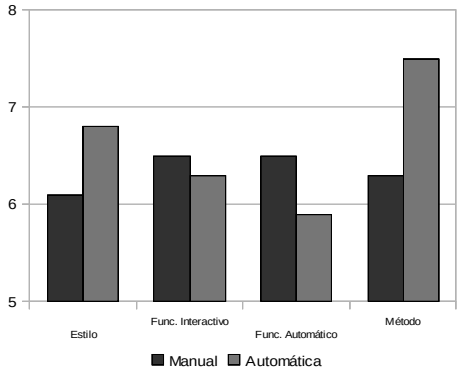

Figura 3. Notas Medias comparadas según concepto de evaluación.

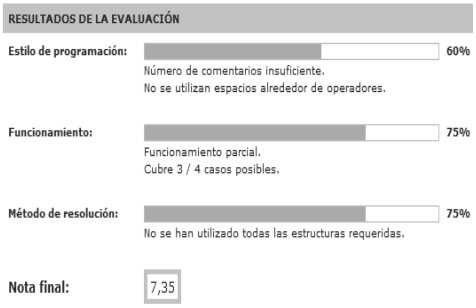

Figura 4. Ejemplo de Mensaje ofrecido al alumno.

5. Conclusiones y Trabajo Futuro

Como se demuestra por el amplio interés existente en la comunidad científica actual, las herramientas de evaluación automática de programas son efectivas no sólo para reducir la carga de trabajo del profesor sino también para ofrecer una guía más cercana y detallada al alumno en su proceso de aprendizaje de cualquier lenguaje de programación.

La herramienta propuesta presenta ciertas novedades con respecto a las existentes en el mercado. Las más destacables son su diseño modular y configurable, la independencia con respecto a ciertas características del entorno, el conjunto de técnicas y alternativas que proporciona y la disponibilidad del código fuente.

Durante su desarrollo y aplicación se ha podido observar que ofrece con mayor rapidez los resultados de los ejercicios a un mayor volumen de alumnos, aún sin utilizar herramienta de gestión, así como, la posibilidad de poder evaluar de la forma más desasistida posible enunciados de problemas previamente existentes.

El trabajo futuro más inmediato es ampliar el número de usuarios de la aplicación de gestión para lo cual se establecerá un programa de implantación durante el segundo cuatrimestre del curso 2009-2010. A su vez será necesario ampliar las capacidades evaluadoras de CUESTOR ya que hay factores que aún no cubre con la suficiente solvencia como la innovación en las soluciones, la utilización correcta de identificadores y comentarios, etc.

Referencias

[1] Ahtiainen A., Surakka S., Rahikainen M. *Plaggie: GNU-licensed Source Code Plagiarism Detection Engine for Java Exercises*, Proceedings of the 6th Baltic Sea Conference on Computing Education Research, pp. 141-142, 2006.

[2] Ala-Mutka K., Uimonen T., Järvinen H., *Supporting Students in C++ Programming Courses with Automatic Program Style Assessment*, Journal of Information Technology Education, Vol. 3, 2004.

[3] Baeza-yates R., Navarro G., New and Faster Filters for Multiple Approximate String Matching, Random Structures and Algorithms (RSA), 1998.

[4] Berry, R.E., Meekings, B.A.E. *A style analysis of C programs*. Communications of the ACM, *28*,pp. 80-88. 1985.

[5] Castillo, P. A.; Cañas, A.; Castillo-Valdivieso, J.J. y Prieto, A. *Sistema web de detección de copias en prácticas de programación*. Actas XII Jornadas de Enseñanza Universitaria de la Informática

[6] http://smile.esi.uclm.es:8080/hugin

(JENUI'06), pp. 519-526, Deusto (Bilbao), España, 2006

[6] Clemente, P.J.; Gómez, A. y González J. *La copia de prácticas de programación: el problema y su detección*. Actas X Jornadas de Enseñanza Universitaria (JENUI'04) pp- 204-210. Alicante, España. 2004.

[7] Higgins, Colin A. and Gray, Geoffrey and Symeonidis, Pavlos and Tsintsifas, Athanasios, *Automated assessment and experiences of teaching programming*, Journal on Educational Resources in Computing (JERIC), ACM Press, Vol. 5 N. 3, 2005

[8] Marco Galindo, M.J., Prieto Blázquez J. *Necesidades específicas para la docencias de programación en un entorno virtual.* Actas de las VIII Jornadas de Enseñanza Universitaria de Informática, JENUI 2002, pp. 5 – 12, Cáceres, Julio 2002.

[9] Martín-Corena R., Albillos I., Rebollo J., López C.. *Detección de copias en prácticas de programación con Jdup*, Actas XIV Jornadas de Enseñanza Universitaria de la Informática (JENUI'08), 2008.

[10] Prechelt L., Malpohl G., Philippsen P. *Finding plagiarisms among a set of programs with Jplag* Journal of Universal Computer Science, March 28, 2000

[11] Redish, K.A., & Smyth, W.F. Program style analysis: A natural by-product of program compilation. Communications of the ACM, 29, 126-133, 1986.

[12] Rodríguez J.C, Díaz Roca M., Hernández Z. Gonzalez J.D. *Hacia la Evaluación continua Automática de Prácticas de Programación*, Actas de las XIII Jornadas de Enseñanza Universitaria de Informática, JENUI 2007, Teruel, 2007.

[13] Zin A., Foxley E., *Automatic Program Quality Assessment System*, Proceedings of the IFIP Conference on Software Quality S P University, Vidyanagar, India, 1991.