Detección de la Copia de Prácticas de Programación con JDup

Raúl Marticorena[1], Ismael Albillos[2], Jonathan Rebollo[2] y Carlos López[1]

[1]Área de Lenguajes y Sistemas Informáticos
Depto. Ingeniería Civil
Universidad de Burgos
{rmartico,clopezno}@ubu.es

[2]Universidad de Burgos
{iba0015,
jrg0021}@alu.ubu.es

Resumen

Un problema bien conocido a la hora de evaluar a los alumnos, en el ámbito de la programación, es lo fácil y habitual que resulta la copia de la prácticas. Si se quiere realizar una evaluación justa y objetiva, el docente debe disponer de herramientas que faciliten la labor de separar las prácticas originales, fruto del trabajo de los alumnos, de aquellas entregas, fruto de la mera acción del plagio o copia.

Con este objetivo, se presenta y describe en este trabajo una herramienta que facilita la labor de detección de copias a través de indicadores de similitud. Trabajando con un volumen notable de entrega de prácticas, y con unos recursos mínimos, habilita de una manera amigable la detección y la generación de informes de similitud de prácticas. Esto permite un análisis más completo por parte del docente, para detectar y tratar de solucionar este problema.

1. Introducción

La copia de prácticas es un problema a la hora de evaluar, bien conocido en el campo docente. Pese a todos aquellos razonamientos en su contra que se puedan hacer a los alumnos, desgraciadamente dicho problema surge siempre.

La simple confianza en el hecho de que no se producirán copias en las entregas de las prácticas, es utópico. Conclusiones de este tipo ya han sido establecidas [3] [14]. El volumen y número de las entregas, junto con la distancia en el tiempo y la impartición de la asignatura por varios docentes, dificulta en gran medida la detección de estas situaciones.

Por lo tanto, la solución más racional, en un contexto informático, es utilizar herramientas que ayuden a pelear contra esta plaga, automatizando dicho proceso.

Este trabajo presenta una aplicación que, en la línea de otras herramientas, se muestra como un elemento adicional en la ayuda al docente a erradicar esta situación, añadiendo alguna característica particular que la hace especialmente adecuada.

En lo que sigue, la Sec. 2, presenta nuestra experiencia particular previa en el desarrollo de este tipo de herramientas y motivación, presentando en la Sec. 3 el estado del arte. La Sec. 4 presenta una visión general de sus características, para posteriormente, en la Sec. 5 mostrar características particulares, el flujo de trabajo habitual con la herramienta y algunos de los resultados obtenidos. Se concluye en la Sec. 6 con las conclusiones y líneas de trabajo futuro.

2. Desarrollos Previos

En años previos se desarrolló una herramienta que permitía el envío y detección a través de la web, similar a sistemas como JPlag [12] o MOSS [1]. Este trabajo se desarrolló en .NET (Visual Basic .NET) integrado con un parser Java y una interfaz en la Web con ASP.NET alojadas en Internet Information Server y base de datos SQL Server. La herramienta incorporaba varios resultados de otras herramientas (PMD [6] y Simian [13]) así como un algoritmo basado en características estructurales de código, para comparar resultados.

Aunque el sistema implementado funcionaba correctamente, para su explotación surgieron distintos problemas. Los recursos hardware y software para instalar una aplicación de este tipo son relativamente altos. En este primer desarrollo era necesario un mínimo de dos aplicaciones de tipo servidor (web y base de datos), dos frameworks de ejecución (.NET y Java) y algún producto adicional de terceros.

El equipo donde se instala requiere además un cierto mantenimiento, teniendo que

responsabilizarse alguien de la gestión y control de usuarios. Otras herramientas de este tipo también requieren de una instalación de frontales adicionales [1], creación de cuentas, gestión de tickets, mantenimiento de equipo, etc.

Aunque a priori esto puede parecer trivial, en la práctica surgen más problemas de los inicialmente planteados, que pueden provocar un cierto abandono de su uso.

Por lo tanto se requería que la nueva aplicación fuese simple de instalar, multiplataforma y que requiriese pocos recursos para su uso. Estas características permiten a cualquier usuario poder obtener resultados en su ordenador personal, y llevar a cabo análisis más particulares mediante su parametrización.

El motivo de plantear una nueva herramienta de detección de código duplicado viene también condicionado por el esquema particular de la entrega de prácticas en primeros cursos, extrapolables a otros contextos universitarios.

A los alumnos se les entrega un diseño y códigos parciales (fundamentalmente especificaciones en la forma de interfaces), que deben ser implementados. Es un hecho fundamental el que los alumnos respeten los nombres y signaturas (contratos) puesto que se suele aplicar a posteriori pruebas automáticas.

Esto se refuerza con la incorporación de bibliotecas entregadas por los docentes que incorporan interfaces gráficas, con las cuáles deben enlazar de manera dinámica. Los productos fuente, binarios y documentación, junto con la automatización de las tareas (scripts) que permitan regenerar todos los productos a partir del código fuente se entregan en un fichero comprimido (formato zip).

Por lo tanto, se tienen un par de características particulares en las entregas:

- diseño estricto que debe ser respetado por el código.
- entrega en formato comprimido.

3. Trabajos Relacionados

Herramientas habituales para la comparación de ficheros con comandos como diff o herramientas como Kdiff3 [8], están limitadas a comparaciones de texto estricto. En la primera versión, realizada en web, se pudo constatar como el análisis de estructura mejora resultados frente a comparaciones literales línea por línea, como los que también se realizan en PMD [6] o Simian [13].

En [2] se presentó una herramienta similar basada en web. Denominada *Sistema de Detección de Copias* (SDC) como frontal (front-end) de [1]. Dicha aplicación ofrece una funcionalidad similar y un mayor número de lenguajes, pero limita las posibilidades respecto a los datos ofrecidos por MOSS. En [12], se presenta JPlag, otra herramienta similar, que permite comparaciones de estructura, pero con los mismos pros y contras que MOSS.

En la misma línea del trabajo aquí presentado, en [3] se describe brevemente el sistema HERACLES, similar en teoría a la solución propuesta, pero no se ha podido disponer de una copia de dicho producto para contrastar parecidos y diferencias. Otra herramienta a señalar es AC [10] con ejecución local, generación de informes y multilenguaje.

Siendo conscientes de la gran cantidad de herramientas que existen en esta línea, se ha intentado dar otro punto de vista al problema, aportando diferentes utilidades que faciliten la detección frente a las herramientas previamente referenciadas.

4. Características generales

La herramienta, denominada JDup, está basada en la detección de porciones de código duplicado, utilizando una detección basada en estructura, con el algoritmo de *greedy string tiling* [16][17] y *pattern matching* de Karp-Rabin [11]. A partir de un análisis léxico se extraen los tokens de interés para posteriormente buscar coincidencias.

El sistema es independiente del lenguaje sobre el que se buscan duplicados, en tanto en cuanto se disponga de la gramática que permita la generación del analizador correspondiente. En nuestro caso concreto se trabaja con gramáticas para JavaCC [7] , lo que limita el número de lenguajes incorporados.

En la versión actual se dispone de una implementación de Java con una gramática válida para código a partir de la versión 1.5. La corrección del parser se ha validado con las prácticas entregadas en el anterior y presente curso, donde se utilizan aspectos recientes del

lenguaje como tipos enumerado, `for` mejorados, genericidad, etc.

También se dispone de una implementación del parser para C#, validado con códigos de prueba, y se incluye un parser para C y C++, aunque la validación sólo se ha realizado con código C de la asignatura de primer curso de fundamentos de programación.

Dado que el algoritmo no depende del lenguaje, sino de la recolección correcta de tokens, la herramienta ha sido probada con altas cargas sobre las prácticas realizadas en Java, pudiendo generalizar su funcionamiento correcto con otros lenguajes, dependiendo sólo de la corrección de la gramática y parser generado.

5. Características particulares

La aplicación sólo admite como entradas ficheros comprimidos. Como se comentó previamente, surge de la particularidad de las entregas. Este sistema se ha utilizado a lo largo de cursos previos, y ha demostrado ser adecuado por mantener una escala manejable. Además no es complicada la generación de scripts que compriman directorios, y la propia aplicación se ha dotado de utilidades para la compresión y descompresión.

La aplicación trabaja a partir de dichos ficheros, con unos conceptos particulares que deben ser presentados al usuario que inicia su uso:

- Proyecto: agrupación de ficheros comprimidos ligados a un lenguaje particular (Java, C# , C, C++, etc.). Un proyecto contiene ficheros comprimidos (equivalentes a una práctica) de los que sólo se toman aquellos ficheros con extensión correspondiente al lenguaje fuente seleccionado.
- Análisis: proceso de detección de duplicados sobre los ficheros comprimidos del proyecto. A lo largo del tiempo se realizarán distintos análisis modificando los argumentos como: cota mínima para la longitud del número de tokens coincidentes, ficheros comprimidos utilizados o criterios de análisis a realizar.

5.1. Criterios en el análisis

Básicamente se permiten dos criterios a la hora de establecer comparaciones entre los códigos fuente contenidos en distintos ficheros comprimidos:

1. Convención de nombres: se comparan sólo aquellos ficheros que coinciden en nombre, evitando comparar ficheros sin relación.
2. Todos contra todos: se aplica fuerza bruta comparando todos los ficheros de un fichero comprimido, con todos los del segundo.

El primer método es el elegido para la revisión de las prácticas por las características planteadas previamente en la Sec. 2. A nuestro juicio, en primeros cursos, acotar las posibles soluciones de diseño facilita la labor tanto del alumno como del docente.

El método de fuerza bruta añade una complejidad de cómputo adicional, requiriendo una máquina con muchos más recursos, y por las características de los enunciados utilizados, no ha sido tan utilizada.

5.2. Indicadores calculados

Para cada comparación entre prácticas, se asocian en la actualidad dos indicadores de similitud:

1. Similitud respecto al fichero más pequeño (Sim_1): número de tokens coincidentes dividido por el número de tokens del fichero más pequeño en la comparación. Posteriormente se realiza una suma ponderada de dicho valor en cada comparación parcial, para obtener el valor total del fichero comprimido. Permite detectar situaciones en las que el plagiador añade nuevas líneas al código inicial. Se considera que el fichero de mayor tamaño ha sido "inflado" respecto al fichero de menor tamaño, intentando introducir ruido en el análisis, por lo que se prefiere el tamaño menor entre los dos ficheros.

$$Sim_1 = \frac{\sum |coincidencias|}{\min(|A|, |B|)} \quad (1)$$

2. Similitud de los dos ficheros (Sim_2): calcula la relación entre el doble del número de tokens coincidentes entre los dos ficheros, dividido por la suma del número de tokens de los dos ficheros. Se realiza una suma de dicho valor de cada comparación parcial para obtener el valor del fichero comprimido.

$$Sim_2 = \frac{2 * \sum |coincidencias|}{|A| + |B|} \quad (2)$$

Aunque ambos indicadores ofrecen una medida de similitud, se elige Sim_1 como primer criterio de ordenación, por entender que elimina el ruido que haya podido generar un plagiador con cierta pericia. A partir de estos valores se pueden realizar ordenaciones (rankings) de similitud entre pares de prácticas entregadas (se multiplica por 100 para representarlos en una escala de porcentajes de similitud). Valores altos reflejan excesivos parecidos, que deben ser revisados de manera más detenida por el docente.

5.3. Flujo de trabajo

El flujo de trabajo habitual se describe a continuación. En primer lugar se crea un nuevo proyecto, como se muestra en la Figura 1, indicando los directorios donde alojar los resultados y como punto fundamental, el tipo de código fuente que va a ser analizado (actualmente se puede elegir entre Java, C#, C y C++).

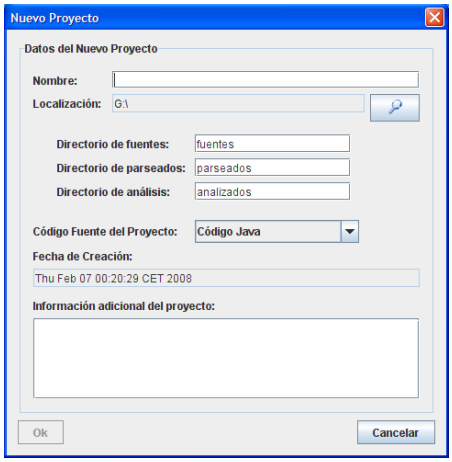

Figura 1. Creación de un proyecto

Una vez creado el proyecto, se deben añadir los ficheros comprimidos, donde cada fichero se corresponde con una práctica. Esto se puede realizar cómodamente a través de opciones del menú, o menú emergente sobre el árbol.

Una vez añadidos, se puede invocar a la generación de análisis, configurando la longitud mínima de coincidencias (MML), el criterio de análisis (convención de nombres o todos contra todos) y elegir los ficheros a comparar, como se muestra en la Figura 2 y Figura 3.

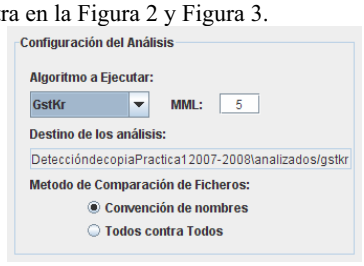

Figura 2. Personalización del análisis

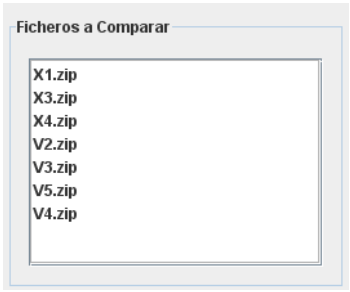

Figura 3. Selección de prácticas

Generado el análisis, éste se puede seleccionar sobre el árbol en la parte inferior izquierda, o bien en los cuadros desplegables de la pestaña de análisis. Al cargar los datos del análisis, se puede seleccionar y visualizar las comparaciones obtenidas entre los diferentes ficheros como se indica en la Figura 4, cargando la tabla comparativa de los ficheros fuente con su correspondiente indicador de similitud como se muestra en la Figura 5.

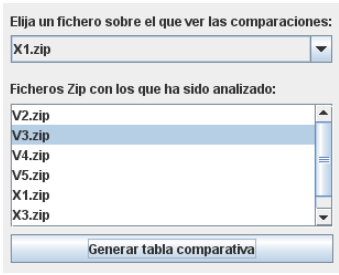

Figura 4. Selección de dos ficheros comprimidos

File	File 1	Zip (file 2)	File 2	%
22.out	FabricaAccesoDat...	V3.zip	FabricaAccesoDat...	100
40.out	VisorImagen.java	V3.zip	VisorImagen.java	96
46.out	Comando.java	V3.zip	Comando.java	95
28.out	FachadaTablaFoto...	V3.zip	FachadaTablaFoto...	86
34.out	Fotografia.java	V3.zip	Fotografia.java	81
3.out	FabricaBD.java	V3.zip	FabricaBD.java	79
17.out	FachadaTablaFoto...	V3.zip	FachadaTablaFoto...	74
52.out	PBarraDesplazami...	V3.zip	PBarraDesplazami...	71
9.out	PDatos.java	V3.zip	PDatos.java	69
13.out	Mediador.java	V3.zip	Mediador.java	60

Figura 5. Tabla de comparaciones

En caso de que el usuario desee ver de manera más detallada las coincidencias entre dos ficheros de código fuente, se permite cargar la comparación en la pestaña de duplicados, para visualizar en color rojo las líneas donde se detectan bloques de tokens similares. El coloreado por líneas provoca que la herramienta coloree más tokens de los que realmente coinciden, siendo éste un punto a mejorar en futuras versiones.

5.4. Informes generados

Construir la herramienta desde cero, permite la incorporación de nuevos elementos, como en este caso la posibilidad de generar diferentes tipos de informes:

- Informe de análisis: genera una tabla ordenada de mayor a menor con todas las comparaciones entre prácticas, sin repetir emparejamientos. Se muestra un ejemplo de informe generado en la Figura 6, en formato HTML. Se detallan los ficheros comprimidos comparados, las características del análisis realizado y los valores de similitud para todos los posibles emparejamientos, ordenados de mayor a menor.
- Informe de tabla comparativa: genera una tabla con los ficheros contenidos en dos ficheros comprimidos. Cada fila representa la comparación entre dos ficheros fuente En la versión HTML se genera un indicador gráfico de dónde y cómo se encuentran las coincidencias en cada comparación. En el eje X se representa el número de línea en el primer fichero donde ocurre una coincidencia, y en el eje Y lo mismo respecto al segundo fichero. A partir de las coordenadas de dicho punto (X,Y) se dibuja una circunferencia cuyo diámetro es el número de tokens coincidentes. Aunque el indicador no es del todo exacto, ofrece una visión rápida del tipo de "copia". En la Figura 7 se observa una copia literal entre dos ficheros. En la Figura 8 se observa una copia por múltiples coincidencias, encontrándose las coincidencias (centros de las circunferencias) sobre la diagonal, lo cual indica que además el orden de los trozos copiados se mantiene en ambos ficheros. Mientras que en la Figura 9 se muestran dos ficheros con coincidencias mínimas y casuales, no encontrándose las coincidencias en las mismas posiciones, por encontrarse fuera de la diagonal principal.

Número de combinaciones: 21

Ranking	Fichero1	Fichero2	Similitud 1	Similitud 2
1	V3.zip	V4.zip	89,33%	84,00%
2	V5.zip	V4.zip	88,89%	77,00%
3	V2.zip	V3.zip	87,30%	82,20%
4	X3.zip	V3.zip	87,00%	79,90%
5	V3.zip	V5.zip	86,91%	78,09%
6	X4.zip	V5.zip	86,71%	70,86%
7	X3.zip	X4.zip	85,43%	70,86%
8	X3.zip	V4.zip	85,33%	76,89%
9	X1.zip	V5.zip	84,80%	71,90%
10	X1.zip	X3.zip	82,20%	72,00%
11	X3.zip	V2.zip	81,90%	74,30%
12	X3.zip	V5.zip	81,80%	75,30%
13	X1.zip	V3.zip	81,10%	73,20%
14	V2.zip	V4.zip	80,50%	74,70%
15	X4.zip	V4.zip	79,50%	73,63%
16	X1.zip	X4.zip	79,29%	73,71%
17	X4.zip	V3.zip	79,14%	69,86%
18	X1.zip	V4.zip	78,89%	74,56%
19	X1.zip	V2.zip	78,67%	71,22%
20	X4.zip	V2.zip	77,14%	67,14%
21	V2.zip	V5.zip	76,90%	71,10%

Figura 6. Informe de análisis

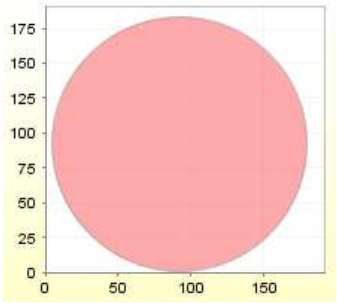

Figura 7. Fichero copiado

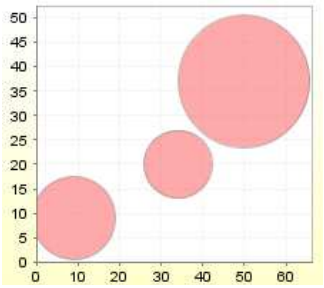

Figura 8. Ficheros copiados con alguna ligera modificación

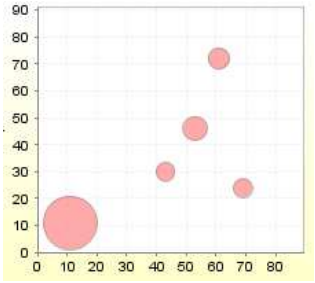

Figura 9. Ficheros con pequeñas coincidencias y en desorden

Un punto fuerte de la herramienta es la generación de este tipo de informes en formato HTML, personalizable su aspecto mediante el uso de hojas de estilo en cascada (CSS).

Por otro lado, también se generan los resultados en formato XML. Los ficheros XML generados se pueden importar en herramientas como hojas de cálculo, permitiendo análisis mucho más detallados de los resultados iniciales, y la generación de otro tipo de informes a través de herramientas ofimáticas. La conversión de XML a formatos como csv para su análisis en herramientas de minería de datos queda como línea abierta de trabajo.

5.5. Experimentos Realizados

Se ha utilizado la herramienta con las prácticas entregadas en este curso en la asignatura de programación orientada a objetos, donde se ha utilizado Java como lenguaje de programación. Se han solicitado dos prácticas obligatorias en el cuatrimestre.

La primera entrega está formada por 60 prácticas (ficheros comprimidos), con un total de 10 ficheros fuente (extensión .java) en cada uno. La segunda se ha reducido a 50 prácticas con un total de 13 ficheros fuente en cada uno.

La aplicación realiza comparaciones de todos los ficheros comprimidos entre sí, no repitiendo aquellos emparejamientos ya realizados (en torno a unas 1770 parejas en el primer caso y 1275 en el segundo). Se ha aplicado el análisis por convención de nombres, realizando aproximadamente 17000 y 16500 comparaciones respectivamente. El número se reduce por aquellas prácticas que no han respetado los nombres y signaturas marcadas, prácticas que no compilan correctamente, etc.

La herramienta ha permitido detectar un número de copias de un 12% sobre el total de trabajos entregados en la primera práctica. Los porcentajes de similitud en la primera entrega entre [90,100] exponen la existencia de demasiadas coincidencias revelando copias. Los valores entre [85,90) también han sido revisados realizándose una revisión con los alumnos para constatar que no existe copia. Estos valores, son dependientes de la práctica concreta y de las características de la misma, observándose que en la segunda práctica el número de prácticas por encima del 90% era inexistente, por lo que es necesario ajustar los umbrales.

Otro punto de interés es la detección inmediata de aquellos alumnos que no respetan lo indicado en el enunciado, respecto a los nombres a utilizar, puesto que no se obtienen valores numéricos en sus comparaciones.

En la segunda entrega el número de copias detectadas se ha reducido drásticamente. Aunque a los alumnos no se les ha explicado abiertamente que se estaba haciendo uso de la aplicación, sí que han percibido que existía un cierto control por parte de los docentes sobre el total. Esto ha evitado las típicas tentaciones en las siguientes entregas de copiar a otro compañero, para ver si se pasaba el trámite.

5.6. Consideraciones de Rendimiento

La aplicación JDup se ha utilizado en modo monousuario, con un procesador Pentium 4, a 3.00GHz, con 384 MB de RAM. El proceso de análisis de 60 prácticas con unos 10 ficheros de tamaño pequeño/medio, tarda en torno a 30 minutos, siendo este el proceso más largo que se realiza en la herramienta.

El resto de procesos, consultas, generación de informes globales o parciales, etc. se puede realizar en tiempo real con pequeños tiempos de espera de segundos. Es este uno de los puntos a mejorar, sobre todo por la alta carga que genera el trabajo con muchos ficheros de tamaño muy pequeños, como son los ficheros de trozos duplicados.

A nuestro juicio los tiempos son aceptables para la ejecución de la herramienta en ordenadores personales, que no tengan altos rendimientos, lo cual era uno de los objetivos iniciales.

5.7. Consideraciones Legales

Un aspecto no siempre considerado a la hora de abordar la creación de un recurso docente, son las cuestiones relacionadas con la viabilidad legal del producto a liberar.

En este recurso se ha optado por utilizar una licencia Creative Commons [4], que permite su uso, copia y distribución, pero por otro lado exige reconocimiento de la obra, sin beneficio económico y sin obras derivadas. Aunque este tipo de licencia, está más orientada a contenidos web, a juicio de los autores nos ha parecido la más adecuada, puesto que permite a posteriori relajar las condiciones, con aprobación de los mismos.

El uso de este tipo de licencias es viable puesto que todas las bibliotecas contenidas y enlazadas, son bibliotecas bajo licencias de tipo LGPL, NPL, o bien Apache License, que permiten este tipo de distribución, y cuyo código fuente no ha sido necesario modificar. Este punto se recoge con mayor detalle en el menú de ayuda de la herramienta, en la opción “Acerca de”.

6. Conclusiones y Líneas de Trabajo Futura

Como se ha comentado previamente, las primeras experiencias con la herramienta han sido exitosas. Sin embargo, la herramienta no libera de tener que revisar el código que es sospechoso de copia. Ciertas similitudes parciales siempre pueden producirse. Más aún cuando el enunciado está muy cerrado, y en el laboratorio se han dado pistas comunes que todos los alumnos aplicarán. Además, si se incentiva el trabajo en grupo, la discusión y el intercambio de opiniones en el laboratorio, el número de coincidencias puede aumentar.

Anteriormente la detección de copias era complicada de localizar y algunos alumnos se aprovechaban de dicha situación realizando algunos ligeros cambios en el código, sabiendo que una comparación manual es inviable e incluso un mismo profesor puede no percatarse si las prácticas se entregan en distintos grupos con cierta distancia en el tiempo. Esta situación sí que se ha conseguido eliminar.

Otro de los resultados indirectos es la detección de alumnos con hábitos de programación muy diferenciados respecto al resto de compañeros. Este punto se ha observado con prácticas de alumnos del programa de intercambio Erasmus, así como con la propia solución de los profesores. Los grados de similitud en estos casos fueron bajos con respecto al resto de prácticas. El perfil particular puede influir a la hora de programar y en la solución final aportada. Este punto queda sujeto a trabajos posteriores, porque la herramientas podría permitir detectar patrones y hábitos de programación en grupos.

Desde un punto de vista general, y como también se comenta en [1] cruzar y comparar los resultados obtenidos con otras herramientas permite establecer con mayor determinación la bondad de la herramienta. En nuestro caso, para la calificación de las prácticas se aplican pruebas de caja negra y medida de la cobertura de las mismas. Las medidas obtenidas de cobertura y complejidad (a través de JUnit [9] y cobertura [5] en las prácticas realizadas en Java), nos permiten cruzar los datos obtenidos. La inclusión de los valores obtenidos con éste u otros sistemas similares, queda como línea de trabajo futura.

Desde otro punto de vista, más particular a la propia herramienta, se pueden señalar como mejoras funcionales a incluir:

- Generar informes en otros formatos (pdf, rtf, etc.)
- Obtener un mayor grado de personalización de los informes.
- Permitir el filtrado (inclusión/exclusión) de ficheros.
- Incluir nuevos algoritmos y comparativas de resultados.
- Mejorar rendimientos.
- Ampliar el número de lenguajes soportados.

En este trabajo, no se ha intentado establecer que las soluciones basadas en web no sigan siendo

válidas, sino que otro tipo de soluciones pueden ser complementarias, y dar una mayor comodidad al docente para la detección de copias. Para reafirmar la importancia de los sistemas en web, en la actualidad también se trabaja en una evolución de la aplicación mediante el uso de servicios web, utilizando el interfaz gráfico como frontal frente a servidores.

Finalmente, indicar que se han conseguido los dos objetivos básicos perseguidos en este trabajo: reducir la copia de prácticas de programación y facilitar la labor del docente. Para una evaluación más detallada, la aplicación está disponible en http://pisuerga.inf.ubu.es/rmartico/JDup.

Agradecimientos

Este trabajo ha sido posible gracias a los esfuerzos previos de Lorena Gil y Raquel Pastor, que desarrollaron una versión previa, similar a [12] en .NET y Java, con un entorno web basado en ASP.NET. Aunque en su mayoría el código ha sido reescrito en Java, muchos de sus resultados y experiencias, han podido ser reutilizados para la elaboración del presente recurso docente.

Referencias

[1] Aiken, A. *MOSS. A System for Detecting Software Plagiarism*. http://theory.stanford.edu/~aiken/moss/. Última visita a 5 de enero de 2008.

[2] Castillo, P. A., Cañas, A., Castillo-Valdivieso, J. J. y Prieto, A. *Sistema web de detección de copias en prácticas de programación.* Recurso docente en Actas XII Jornadas de Enseñanza Universitaria de la Informática (JENUI'06), pp 519-526, Deusto (Bilbao), España, 2006.

[3] 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 de la Informática (JENUI'04), pp 204-210, Alicante, España, 2004.

[4] Creative Commons. *Creative Commons España.* http://es.creativecommons.org/. Última visita 5 de febrero de 2008.

[5] Doliner, M. *Cobertura.* http://cobertura.sourceforge.net/ Última visita realizada a 4 de febrero de 2008.

[6] InfoEther, Inc . *PMD.* http://pmd.sourceforge.net/ Última visita 3 de febrero de 2008

[7] JavaCC. *javacc; JavaCC Home.* https://javacc.dev.java.net/. Última visita a 5 de enero de 2008.

[8] Joachim, E. KDiff3. *Text diff and merge tool*. http://kdiff3.sourceforge.net. Última visita 3 de febrero de 2008

[9] JUnit. *Testing Resources for Extreme Programming.* http://www.junit.org. Última visita 26 de enero de 2008.

[10] Freire, M., Cebrián, M. and Rosal, E. *AC. An anti-plagiarism system for programming assignments.* http://tagow.ii.uam.es/ac/. Última visita 22 de abril de 2008.

[11] Karp, R.M & Rabin, M.O. *Efficient randomized pattern-matching algorithms.* IBM J. of Research and Development. 31(2):249-260, 1987.

[12] Malpohl, G. & Cocheteau, M. *JPLAG. Detecting software plagiarism.* https://www.ipd.uni-karlsruhe.de/jplag/. Última visisa 5 de enero de 2008.

[13] RedHill Consulting Pty. Ltd. *Simian : Similarity Analyser, v 2.2.21*. http://www.redhillconsulting.com.au/products/simian/. Última visita 3 de febrero de 2008

[14] Sheard, D., Markahm, MacDonald y Walsh. *Cheating and Plagiarism: Perceptions and Practices of First Year IT Students*. ITiCSE, 2002.

[15] Sun Microsystems. *Java 2 Platform Standard Edition 6.0.* http://java.sun.com. Última visita 3 de febrero de 2008

[16] Wise, M.J. *Detection of similarities in student programs: YAP'ing may be preferable to Plague'ing*. ACM SIGCSE Bulletin (Proc. of 23rd SIGCSE Technical Symp.), 24(1):268-271, March 1992.

[17] Wise, M.J *String similarity via greedy string tiling and running Karp-Rabin matching.* Dept. of CS, University of Sydney. 1993.