2017

SISTEMA SEMI-AUTOMATIZADO DE CORRECCIÓN DE TAREAS DE PROGRAMACIÓN PARA EL DEPARTAMENTO DE INFORMÁTICA DE LA UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

CUADRA HERRERA, FRANCISCO ESTEBAN

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

VALPARAÍSO – CHILE

SISTEMA SEMI-AUTOMATIZADO DE CORRECCIÓN DE TAREAS DE PROGRAMACIÓN PARA EL DEPARTAMENTO DE INFORMÁTICA DE LA UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

Francisco Esteban Cuadra Herrera

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN INFORMÁTICA

PROFESOR GUÍA: Dr. Horst von Brand

PROFESOR CORREFERENTE: Sergio Campos

JUNIO - 2017

Contenido

Índice de Figuras, Tablas y Ecuaciones:

Agradecimientos:

Aquellos que me conocen saben que no me gusta compartir mis triunfos, pero pensando en todas las personas que me han ayudado a llegar hasta aquí, debo decir que este triunfo no es realmente mío.

Gracias a mis padres, Francisco y Alejandrina, que me dieron la educación necesaria para llegar hasta aquí y se aseguraron que nunca nada me faltara, que me impulsaron a perseguir mis sueños sin importar lo que nadie dijera y que me apoyaron incluso cuando me ponía difícil.

Gracias a mis hermanos, Alejandra, Paulina y Daniel, que estuvieron ahí conmigo para compartir risas y llantos, que me escuchaban cuando me quejaba y me recordaban que tenía lo necesario para salir adelante.

Gracias a mis amigos, Anthony, Eduardo, Milla, Michelle, Cristóbal, Salvador, Samuel, Andrés, José Miguel, Matías, Paulina, Carlos, Jana, Sofía, Koda, Juan, Mauricio y sí, también Felipe. Todos ustedes han hecho de mis éxitos triunfos y de mis fracasos contratiempos.

Gracias a mis profesores, obviamente, porque tratar de sacar un título de ingeniería sin alguien que te enseñe puede ser bastante más lento. Demasiados excelentes profesionales de la enseñanza han hecho esto posible, pero creo que mis gracias van especialmente al profesor Zerwek del departamento de Física, por recordarme que me gusta aprender, al profesor Sven von Brand y sus ayudantes, por su asistencia durante esta memoria y, sobre todo, al profesor Horst von Brand, mi profesor guía y uno de los mejores del DI.

Y por último, pero no menos importante, gracias a Dios, según entiendo no es demasiado popular en esta universidad, pero mirando hacia atrás a todo lo que pudo haber salido mal, realmente siento que lo tuve de mi lado estos años.

"No conozco a la mitad de ustedes, ni la mitad de lo que querría, y lo que yo querría es menos de la mitad de la mitad de lo que ustedes merecen".- JRR Tolkien, El Señor de los Anillos.

Resumen:

En el campo de las ciencias de la computación, las tareas prácticas de programación forman una piedra angular del aprendizaje, sin embargo, la corrección de dichas tareas puede verse influenciada por múltiples factores que perjudican el aprendizaje.

Con esto en mente, se presenta un sistema semi-automatizado de corrección de tareas de programación para el Departamento de Informática de la Universidad Técnica Federico Santa María, el cual es capaz de ejecutar automáticamente tareas en lenguaje C o Python, dado que dichas tareas reciban datos de entrada vía archivos de texto y entreguen datos de salida por el mismo método. El sistema puede, además, evaluar la correctitud de los datos de salida generados, indicar la presencia o ausencia de archivos auxiliares (tales como bibliotecas propias o makefiles), indicar uso de comentarios e indentación y comparar tareas entregadas en busca de plagio.

Palabras Clave: Corrección Automática, tareas, programación, C, Python.

Abstract:

In the field of Computer Sciences, practical homework constitutes a corner-stone of learning, however, a crucial task such as the assessment of said homework may be affected by multiple factors, which harm the quality of the learning.

With this in mind, the following system is presented: a semi-automated assessment system for programming assignments for the Departamento de Informática of Federico Santa María Technical University, capable of automatically running programs in the C or Python languages, given said programs receive input data from text files and produce output data via the same method. The system can, additionally, assess the correctness of produced output, indicate the presence or absence of auxiliary files (such as the student's own libraries or makefiles), indicate the use of comments and indentation and compare submissions in search of plagiarism.

Keywords: Automatic Assessment, assignment, homework, programming, C, Python.

Capítulo 1: Introducción.

En la gran mayoría de las áreas de la ingeniería, los proyectos son desarrollados con una rigurosidad mucho mayor a la que se da en informática, algunos incluso dirían que si se construyeran puentes o edificios con la misma rigurosidad y meticulosidad con la que se desarrolla software, habría muchos más accidentes y desastres. El software con errores o *buggy* es una espina en el costado de todo desarrollador, una que se hace presente en los cientos de bugs descubiertos por usuarios finales una vez lanzado el programa. Una de las mejores formas de atender esta situación es mejorando la educación del futuro desarrollador: inculcando buenas prácticas desde el aprendizaje más básico. Las tareas forman una parte vital de este aprendizaje, por lo que su corrección también. Este último proceso, sin embargo, puede ser largo, tedioso y, dependiendo del nivel de concentración y energía del corrector, puede ser incluso desigual entre alumnos. Un sistema de corrección automática agilizaría esta tarea y garantizaría la uniformidad de la evaluación.

1.1 Programación y la USM

El Departamento de Informática de nuestra universidad está utilizando Moodle[1] como su plataforma principal de comunicación entre alumnos y profesores, a través de esta se manejan asuntos como calendarios de evaluaciones, entrega de notas, inscripción de grupos de trabajo y, por supuesto, entrega de tareas. En el pasado, cuando estas tareas se manejaban a través de Dotlrn[2], existía un sistema que descargaba automáticamente las tareas, desde el cambio a Moodle, este sistema se ha perdido.

[1] https://moodle.org/

[2] http://dotlrn.org/

El Departamento de Informática de nuestra universidad se hace cargo de la docencia del ramo de Programación (IWI-131), el cual es un curso obligatorio para la mayoría de los alumnos de primer año, no sólo de Informática, sino de toda la Universidad, impartiéndose a más de cincuenta paralelos cada año entre Casa Central y los campus Vitacura y San Joaquín (1) Considerando que, para un futuro profesional de la informática, es de suma importancia contar con una sólida base de conocimientos y habilidades generada en los cursos más básicos, es en este curso donde se debe iniciar un esfuerzo para reforzar dichos conocimientos y habilidades.

En la asignatura de Programación, los estudiantes de la UTFSM aprenden las nociones básicas de la programación computacional. El lenguaje de programación utilizado para ello es Python. Las tareas asignadas para realizar fuera de los horarios de clases y ayudantías tienen un fuerte componente gráfico, lo cual las hace bastante diferentes a las tareas de ramos de informática más avanzados como son Estructuras de Datos o Lenguajes de Programación, sin embargo, los controles y ejercicios de programación son bastante similares en enfoque a las tareas de futuros ramos del Departamento de Informática. Habiendo dicho esto, la asignatura con el mayor número de estudiantes y, por lo tanto, la asignatura que representa el mayor trabajo a la hora de corregir tareas es la de Programación. La mayor parte de estos estudiantes provienen de otras carreras.

Un estudiante de Ingeniería Civil Informática de nuestra Universidad, según la malla curricular vigente, cursará once semestres antes de obtener su título. Al considerar la cantidad de estudiantes que ingresan a la carrera y multiplicar esto por la cantidad de ramos y tareas, la corrección de estas últimas pasa a ser una tarea monumental. Para ilustrar el punto anterior, se presenta una figura de la malla curricular vigente, en la cual puede apreciarse que el estudiante promedio del Departamento de Informática tendrá que realizar tareas de programación para, por lo menos, una asignatura por semestre.

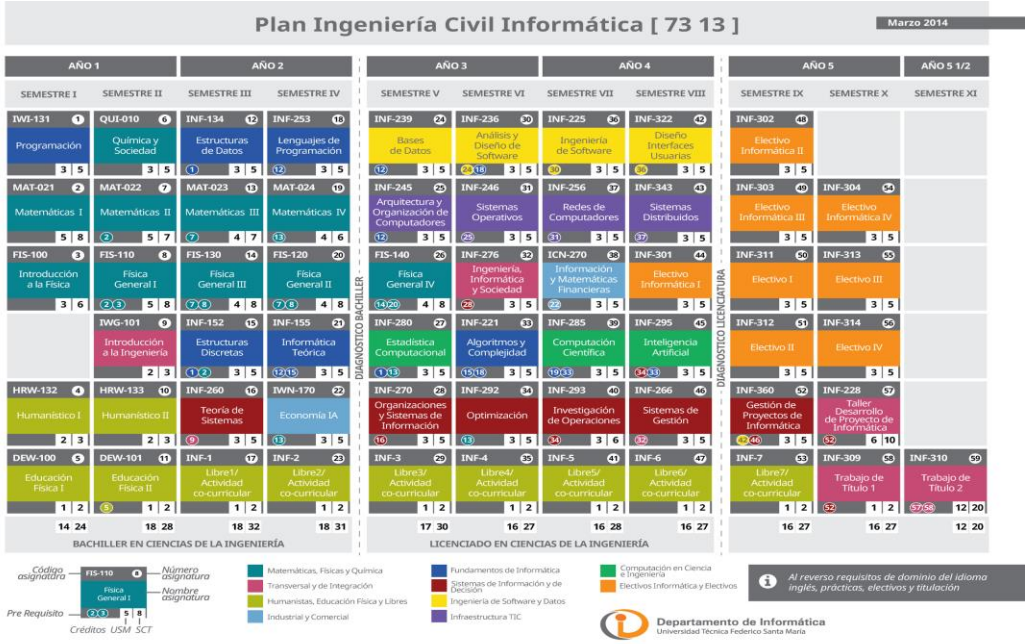


Figura 1.1.1: Malla Curricular vigente de Ingeniería Civil Informática en la UTFSM. Fuente: Departamento de Informática, visitada el día 3 de Febrero de 2017

1.2 Corrección Automática y Enseñanza:

Pieterse (2) lista una serie de factores necesarios para la implementación y uso exitoso de una herramienta de corrección automática, estos son:

- Tareas de Calidad: En un curso cuyas tareas son corregidas manualmente, es posible compensar un pobre diseño de la tarea al otorgar puntaje adicional por una solución creativa, sin embargo, al implementar la corrección automatizada, esto se vuelve más difícil, por lo que la calidad de la tarea misma se vuelve mucho más importante para el impulso del aprendizaje.
- Formulación clara de trabajos: Una máquina no es un ser humano, por lo tanto, para que la corrección automatizada funciona, los objetivos de las tareas deben ser definidos con la mayor precisión posible.
- Datos de prueba bien escogidos: El valor formativo de una tarea depende en gran medida de la detección de errores durante su corrección, con datos de prueba bien escogidos, se asegura que un programa equivocado no pase por uno correcto, esto se vuelve tanto más importante cuando se trata de corrección

automática, ya que la idea es que el corrector no tenga que leer uno a uno el código de cada programa entregado.

- Buena retroalimentación: Una de las experiencias más didácticas es el error, sin embargo, dejar al estudiante con un simple mensaje de que algo salió mal no fomenta ningún aprendizaje. Es necesario que los estudiantes, particularmente los menos experimentados en el área de la programación, reciban abundante y clara retroalimentación con respecto del trabajo que realizaron, con el fin de maximizar su aprendizaje. Proveer de retroalimentación completamente automatizada resulta ser un desafío considerable.
- Entregas múltiples: La capacidad de someter a aprobación una tarea múltiples veces permite al estudiante refinar las habilidades que se busca desarrollar con el trabajo asignado, este factor va de la mano con la entrega de buena retroalimentación.
- Maduración de la capacidad de probar el código: El uso de corrección automática es considerado por algunos como beneficioso para la capacidad del estudiante de probar su propio código (3), razón por la cual muchas herramientas proveen la opción de entregar algunos de los casos de prueba a los estudiantes. Esto aporta a la capacidad del alumno de lograr correctitud y robustez en sus programas.
- Soporte adicional: Por supuesto, la corrección automatizada no puede reemplazar la enseñanza presencial, varios investigadores han concluido que las preguntas respondidas por un instructor fomentan la capacidad del alumno de aprender de forma independiente y pensar de forma reflexiva (4).

Así como hay factores de éxito, también existen ciertos aspectos a considerar antes de implementar una herramienta de corrección automática, a saber:

- Esfuerzo extra: Por la propia naturaleza de la corrección automatizada, se debe dedicar un gran esfuerzo a la construcción precisa de casos de prueba y a la asignación precisa de puntuaciones, ya que desviaciones minuciosas podrían crear grandes inconsistencias. De cierta forma, diseñar una tarea de

programación para ser corregida automáticamente es similar a diseñar preguntas bien hechas de selección múltiple.

- Represión de la creatividad: La mayoría de los sistemas de corrección automática, al requerir datos de salida muy específicos para su corrección, limitan de gran manera las formas en que un estudiante puede dar solución a un problema.
- Fraude académico: Existe una opinión popular entre ciertos investigadores (5) (6) de que al no revisarse el código con ojos humanos, se incrementa la probabilidad de que los alumnos cometan alguna clase de fraude académico, como puede ser (pero no se limita a) el plagio o la entrega de tareas deficientes que, técnicamente hablando, cumplan con los requisitos exigidos por el sistema para una buena calificación (por ejemplo, un programa que no realice las operaciones requeridas, pero que genere los datos de salida esperados para casos de prueba determinados). Es de suma importancia tener medidas de contingencia activas contra este tipo de situaciones.
- Aprendizaje: Un sistema que sea notoriamente más difícil de aprender que el que ya está implementado, o uno en el cual un error pueda causar la pérdida del trabajo realizado, llevará a una gran resistencia al uso del sistema. Es importante que el sistema implementado sea no sólo útil, sino también fácil de usar y entender.
- Seguridad: Un programa corregido por un sistema automático correría en el servidor donde se aloja el sistema, con los privilegios del corrector, lo cual abre la máquina a ataques por parte de alumnos de pobre fibra moral.
- Aplicación limitada: Un sistema automatizado no puede corregir todos los tipos de tareas de programación existentes, programas en los que la interacción del usuario con una interfaz visual forma parte de la evaluación serían imposibles, sólo por nombrar un ejemplo.

Cheang et al (7) se refieren a tres criterios que componen la evaluación de una tarea de programación:

- Correctitud: Si, dada una entrada legal, un programa produce la salida deseada, entonces dicho programa se considera *correcto*. Cheang y su equipo definen la robustez del programa, es decir, su capacidad de manejar una entrada ilegal, como una cualidad menos importante con respecto a la correctitud.
- Eficiencia: Un programa es calificado como eficiente si puede resolver el problema en cuestión sin consumir demasiados recursos. La definición exacta de "demasiados recursos" queda a discreción del evaluador y depende del problema en cuestión.
- Mantenibilidad: Un programa se califica como mantenible si su código es fácil de entender, utilizando, entre otras cosas, comentarios, indentación consistente, programación modular y nombres de variables significativos.

Para el proceso de enseñanza, la prioridad, según Cheang, debería estar en la correctitud del programa, ya que en ella radica la capacidad del software de realizar sus labores en el sentido más básico. En un cercano segundo lugar, la eficiencia del programa, ya que un software correcto, pero que consume demasiados recursos u opera de forma demasiado lenta no puede llamarse realmente exitoso bajo ninguna circunstancia. Bajo estos criterios, podría decirse que la mantenibilidad del programa es el aspecto menos importante, sin embargo, aunque no se le debe dar mayor importancia en el contexto del aprendizaje, se debe recalcar que es vital en el desarrollo de grandes aplicaciones comerciales.

La evaluación de una tarea de programación, según Joy et al (8), se define sobre tres pilares principales:

- Correctitud: El verificar que el programa cumpla con las especificaciones entregadas y responda de forma apropiada a los datos de entrada del usuario.
- Estilo: El código debe ser comprensible y fácil de leer para una persona que no sea el desarrollador original de éste, el orden, legibilidad y los nombres

significativos y pertinentes de variables y funciones forman parte de este pilar de la corrección.

- Autenticidad: Este pilar de la corrección tiene relación con la detección de plagio y otras prácticas deshonestas en la realización de la tarea, el código, vale decir, debe ser auténtico.

De la misma forma Joy define ciertos fundamentos pedagógicos que un sistema de corrección debe reforzar en su operación, tales como el uso de comentarios, la consistencia del estilo de programación, la correctitud del uso del lenguaje y de su estructura, las pruebas a conciencia, el uso de bibliotecas y otras formas de modularización del código, la documentación para el usuario final y la eficiencia en el uso de código al programar. Estos atributos son amplios y no son aplicables a todos los programas, sin embargo, una buena cantidad de ellos pueden ser automatizados en la corrección.

Los beneficios que podrían cosecharse de la implementación de un sistema de corrección automática o, como es el caso que se presenta, semi-automática, son múltiples: Una mayor uniformidad en la corrección de tareas de programación, lo cual aporta a una revisión más justa. Una disminución de la carga de trabajo de profesores y ayudantes que no solamente libera a los correctores para consultas o ayudantías, sino que además acelera el proceso de corrección, esto último permite la asignación de una mayor cantidad de tareas por semestre, las cuales podrían cubrir una menor cantidad de contenido y, de esta forma, contribuir a que los estudiantes arraiguen el aprendizaje necesario con mucha mayor fuerza. La posibilidad de un mejor proceso de prueba de los programas de los alumnos durante la etapa de desarrollo, con la retroalimentación necesaria para refinar su tarea, entre otras cosas.

Un beneficio potencial que probablemente no parezca obvio es la posible disminución de plagio en un curso por la implementación de un sistema de corrección

automatizada, en su publicación, Cheang et al (7) afirman que, en un plazo de dos semestres, la implementación de su sistema de corrección automática en el curso *CS 1102 Data Structures and Algorithms* en la Escuela de Computación de la Universidad Nacional de Singapur, disminuyó las instancias de plagio en dicho curso de un 13.76% en el año 2000, comparado con un 1.38% en 2001.

1.3 Objetivos

Tomando en cuenta la información recientemente expuesta, se toma la decisión de desarrollar un sistema semi-automatizado de corrección de tareas de programación para el Departamento de Informática de la Universidad Técnica Federico Santa María. Para lograr esta tarea, se han definido los siguientes objetivos:

Objetivo General:

Desarrollar una arquitectura segura y escalable que permita la corrección automática de tareas de programación para el Departamento de Informática de la Universidad Técnica Federico Santa María.

Objetivos Específicos:

- Evaluar la Factibilidad de una conexión entre el sistema de corrección y la plataforma Moodle del Departamento de Informática. Implementando dicha conexión de ser posible.
- Desarrollar una aplicación capaz de compilar y ejecutar programas para algún lenguaje de programación por definir, evaluando el desempeño del programa según los datos de salida esperados.
- Evaluar tareas de programación según su apego a las buenas prácticas definidas según el lenguaje de programación elegido.
- Implementar medidas de seguridad que protejan al corrector de código malicioso o incorrecto.

El presente trabajo de titulación se ha dividido en los siguientes capítulos con el fin de mejor desarrollar y explicar el tema:

1. **Capítulo 1: Introducción.** El capítulo presente, en el cual se presenta el problema a solucionar, el contexto específico de la UTFSM y se plantean los objetivos del trabajo.
2. **Capítulo 2: Estado del Arte.** En el cual se aborda el trabajo desarrollado por otros investigadores en las áreas de corrección y evaluación automática, desarrollo sobre Moodle y detección de plagio y código malicioso.
3. **Capítulo 3: Solución Implementada.** En el cual se detalla el sistema creado, su modo de uso y los datos de salida generados por este.
4. **Capítulo 4: Validación.** En el cual se especifican las pruebas a las que se sometió el sistema desarrollado y se realiza un análisis de su desempeño.
5. **Capítulo 5: Conclusiones.** En el cual se detallan los logros alcanzados según los objetivos planteados, se presentan los aprendizajes logrados a través del trabajo y se enuncian las posibles direcciones de un trabajo futuro en el presente tema.

Capítulo 2: Estado del Arte:

Existen múltiples alternativas de software que hacen corrección automática de tareas de programación, la razón de esto es que muchos de los sistemas diseñados son justamente memorias, tesis y otros proyectos similares, por lo que sólo se trabaja en ellos durante lo que dura el proyecto.

2.1 Evaluación Automática

Varios equipos (9) (10) han compilado listas de las herramientas disponibles de evaluación automática para tareas de programación, gran cantidad de las herramientas listadas trabajan con programas en Java, siendo este el lenguaje de programación introductorio más utilizado (9), existen, sin embargo, múltiples herramientas para la ejecución y corrección automática de tareas en lenguajes como C/C++, Python, Assembler, etc.

WEB-CAT:

Eduards y Quiñones (11) desarrollaron la herramienta WEB-CAT, utilizada por el Virginia Tech para la corrección automática de las tareas de programación en dicha institución, cuenta con soporte para tareas en Java y C/C++, una interfaz tipo *WYSIWYG* ("*What you see is what you get*", Lo que ves es lo que obtienes) para comentarios y correcciones al código del estudiante y la capacidad de diseñar pruebas para el código presentado. WEB-CAT es una herramienta *Open-Source* alojada en SourceForge[3] y distribuida bajo la *GNU Public Licence*[4], razón por la cual es la herramienta de este tipo más utilizada. Para la fecha de 29 de marzo de 2017, WEB-CAT se encuentra en su

[3] http://sourceforge.net/

[4] http://www.gnu.org/licenses/gpl-3.0.en.html

versión 1.4.0, y cuenta con una activa comunidad de desarrolladores trabajando en expansiones y adiciones.

BOSS:

Otro sistema popular es BOSS (8), desarrollado por Joy, Griffiths y Boyatt, un sistema web que permite a los estudiantes someter código a corrección, probarlo, y re-someter con revisiones dentro de las fechas límites establecidas, mientras que permite a los correctores diseñar un set de pruebas para el código (que puede ser un súper-set de las pruebas provistas a los estudiantes), detectar plagios entre tareas entregadas, e incluso proporcionar retroalimentación en forma de comentarios en el código del estudiante. BOSS ofrece además la capacidad de corregir el código automáticamente según métricas como la correctitud, eficiencia, organización y comentarios presentes en este.

BOSS puede usarse para corregir programas en los lenguajes C/C++, Java y Prolog. El sistema fue liberado como Código Abierto, bajo la Licencia Pública de GNU, en el año 2005, sin embargo, el código y los enlaces de descarga parecen haber desaparecido para la fecha de 29 de marzo de 2017.

Mailing It In:

El sistema *Mailing It In* (12), desarrollado por Joseph Sant, utiliza una plataforma basada en un cliente de correo electrónico, eliminando la necesidad de enseñar a los alumnos el uso de la herramienta. *Mailing It In* extrae, analiza, compila, ejecuta y corrige las tareas enviadas a la casilla de correo, para luego componer mensajes de correo con retroalimentación con respecto a los errores del alumno, soporta tareas en lenguaje Java. Vale mencionar, *Mailing It In* fue desarrollado como expansión de clientes de correo ya

existentes, siendo exitosamente implementado en Sylpheed[5] y Claws-Mail[6] y se encuentra disponible como un *plug-in* para dichos clientes de correo.

EduComponents:

Amelung et al (13) proponen el sistema *EduComponents,* un sistema modular que permite la corrección de ensayos, evaluaciones de selección múltiple y, por supuesto, la revisión automática de tareas de programación. *EduComponents* es una herramienta de código abierto, programada sobre el sistema *Plone*[7] de manejo de contenido, el cual, a su vez, es de código abierto y programado en lenguaje Python, aunque al tiempo de su publicación, el sistema fue liberado para uso y modificación, una búsqueda realizada el día 30 de marzo de 2017 no ha sido exitosa en encontrar vínculos de descarga o código. *EduComponents* soporta tareas en Python, Haskell, Scheme, CommonLisp y Prolog.

CTPracticals: (14)

El sistema *CTPracticals,* desarrollado por Gutierrez et al, de la Universidad de Málaga*,* resulta de particular interés para el caso de la USM, ya que este sistema, diseñado para la entrega y corrección automática de tareas orientadas a VHDL[8], fue desarrollado como un módulo adicional agregado a Moodle. La aproximación utilizada por el módulo *CTPracticals* es realizar dos revisiones: Una formal, en la cual se revisa que los archivos cumplan satisfactoriamente con convenciones establecidas por el curso (nombres, extensiones, compresión, etcétera), y una segunda revisión funcional, la cual es realizada por un agente externo a Moodle. Este proceso, a pesar de requerir un sistema revisor externo, puede hacerse de forma totalmente automática. CTPracticals se

[5] http://sylpheed.sraoss.jp/en/

[6] http://www.claws-mail.org/

[7] https://plone.org/

[8] https://es.wikipedia.org/wiki/VHDL

encuentra disponible para descargar gratuitamente y es compatible con la versión 1.9 de Moodle. La USM utiliza la versión 2.8.3+ de Moodle, con miras a actualizar a la versión 3.2.2+. El equipo de desarrolladores de CTPracticals se encuentra trabajando en exportar la extensión a las versiones 2.X y 3.X de Moodle.

Fitchfork: (2)

Vreda Pieterse, de la Universidad de Pretoria, presenta el sistema *Fitchfork*, el cual comparte muchas características con los anteriormente descritos. *Fitchfork* puede compilar y ejecutar tareas en C++ y genera reportes tanto para el estudiante como para el instructor, ambos conteniendo detalles de la evaluación. Al momento de la publicación de (2), el sistema Fitchfork no se encuentra disponible para descarga ni compra, ya que es de uso exclusivo de la Universidad de Pretoria, para su propio LMS, el cual también es de su exclusivo uso.

Marmoset (15)

Construida en la Universidad de Maryland, por Spacco y colegas, esta herramienta es capaz de monitorear completamente el progreso de un estudiante a lo largo del curso tomado. Marmoset soporta código de cualquier lenguaje de programación y su arquitectura incluye un servidor web J2EE, una base de datos SQL y uno o más servidores de construcción, usados para mantener un ambiente aislado y seguro, previniendo los efectos de códigos maliciosos.

Marmoset realiza análisis estático y dinámico sobre las tareas entregadas, este último se realiza a través de casos de prueba.

La herramienta se encuentra disponible para descargar y editar a través de su página web[9].

[9] http://marmoset.cs.umd.edu/index.shtml

ECAutoAssessmentBox (16)

Esta herramienta, desarrollada por Amelung et al para la Universidad de Magdeburg, utiliza pruebas dinámicas y soporta programas en Haskell, Scheme, Prolog, Python y Java. Su arquitectura considera tres servidores: el front-end, que actúa como un LMS; el *spooler*, que controla las peticiones, las colas de sumisión y las llamadas al back-end, el cual es el que realiza las evaluaciones, para la comunicación entre estos servidores, se utilizan XML-RPC (*Remote procedure calls*).

La herramienta se encuentra disponible como una expansión del sistema Plone de manejo de contenido, descargable desde su página web[10].

JavaBrat (17)

Una herramienta diseñada como parte de una tesis de Magister en la Universidad estatal de San José. Soporta tareas en Java y Scala. El software de evaluación como tal está desarrollado en lenguaje Java, mientras que se usa PHP para construir un plugin de Moodle. El software consta de tres módulos: El primero es un server de Moodle con un plugin; el segundo, contiene un repositorio de problemas y los evaluadores dependiendo del lenguaje necesario y finalmente el tercero, JavaBrat, el cual tiene una serie de servicios para llamar a problemas o evaluadores.

La aplicación proporciona dos tipos de problemas: Problemas Rápidos que ya vienen parcialmente construidos y que el alumno debe completar y Problemas Completos en los que se espera que el estudiante comience desde cero.

La evaluación propiamente tal se hace llamando a la aplicación LabRat, descrita en más detalle más adelante.

La integración de JavaBrat con Moodle se implementa en base a la creación de una nueva clase de pregunta en Moodle: La *Pregunta JavaBrat*, la cual hace uso del *JavaBrat*

[10] https://pypi.python.org/pypi/Products.ECAutoAssessmentBox/1.4.2

Webservice para recolectar detalles del problema como de la solución propuesta, convirtiendo a Moodle en un intermediario.

La Pregunta JavaBrat permite al profesor crear nuevos ejercicios para que los alumnos resuelvan y agregarlos al directorio de problemas de programación existente. El plugin para Moodle como tal no corrige los trabajos, sino que los envía al servicio web de JavaBrat.

Para la creación del tipo de pregunta JavaBrat, se debe crear una tabla de base de datos *question_ap_labrat***,** para esto, Moodle provee una capa de abstracción llamada XMLDB, la cual permite crear nuevas estructuras en la base de datos sin importar realmente el tipo de base de datos que se esté utilizando, para esto, el desarrollador debe crear su estructura en formato XML y guardarla en un archivo llamado *install.xml,* el cual debe introducirse en la carpeta *plugin_name/db* del plugin. Moodle provee una clase llamada *question_edit_form* que permite crear formularios para la edición de preguntas, los cuales fueron utilizados para la creación de la pregunta tipo JavaBrat.

<u>LabRat:</u> (18)

LabRat automatiza la corrección de tareas de Java, el sistema fue creado por el doctor Cay Horstmann como un apoyo a su libro de texto Big Java, para el curso que él mismo dicta en la Universidad Estatal de San José. El sistema se encuentra disponible para las plataformas Wiley PLUS[11] y Blackboard[12], pero no para Moodle. LabRat compila y ejecuta el código entregado por el alumno, probando la correctitud de los datos de salida contra casos de prueba establecidos, con esta información, el software genera un reporte que puede ser texto plano o HTML. Los lenguajes comprendidos por esta herramienta son Java y Scala.

LabRat tiene dos modos de operación: Completo (*Full program grading*) y Completar Código (*Code Completion*), en el primero, el alumno deberá programar su

[11] https://www.wileyplus.com/about.html
[12] http://lac.blackboard.com/

tarea desde cero, mientras que en el segundo se le provee un esqueleto de código con el cual empezar. LabRat necesita los siguientes parámetros para generar un reporte:

- Modo de operación (Completar Código o Programa Completo)
- Ruta del directorio que contiene los archivos del usuario.
- Ruta del directorio que contiene la metadata del problema.
- Ruta del directorio que contiene la metadata del evaluador, como por ejemplo, casos de prueba y sus especificaciones.

<u>AutoLEP:</u> (19)

Wang et al proponen un sistema basado en pruebas tanto estáticas como dinámicas, las pruebas dinámicas se basan en casos de prueba, evaluando la correctitud de los datos de salida generado por el programa, mientras que las pruebas estáticas se basan en la correctitud de la implementación, evaluando la presencia/ausencia y frecuencia de errores de semántica, malas prácticas, uso de comentarios y si es que la tarea cumple con los requerimientos establecidos por el profesor, por ejemplo, puede darse que la tarea tenga como objetivo que el estudiante aplique el concepto de recursividad, pero un estudiante logre conseguir soluciones correctas sin aplicarla, AutoLEP es capaz de identificar estos errores. La herramienta fue puesta a prueba en un curso básico de programación en C y comprende tareas en dicho lenguaje.

Para sus correcciones estáticas, AutoLEP hace uso de la técnica denominada Evaluación Basada en Similitud Semántica (*Semantic Similarity-Based Grading*) (20) (21), en la cual, se comparan dos programas diferentes, uno entregado por el estudiante como tarea y el otro proporcionado por el profesor como una respuesta correcta. Cada programa es sometido a un proceso de representación y estandarización, el cual genera un Grafo de Dependencias del Sistema (*System Dependece Graph, o SDG*), este grafo tiene la característica de que, para dos programas cualesquiera, si sus *SDG* son iguales, entonces los programas son Semánticamente Equivalentes. El equipo de investigadores define una serie de reglas para la evaluación basada en Similitud Sintáctica:

1. La correctitud semántica de un programa determinado puede ser definida en base a un conjunto de programas modelo, que corresponden a los algoritmos que pueden usarse para resolver el problema en cuestión.
2. La precisión del programa de un alumno puede ser evaluada calculando las similitudes semánticas entre dicho programa y los programas modelo proporcionados por el profesor.
3. Un Grafo de Dependencia de Sistema (SDG) es una representación semántica de un programa. Si dos programas tienen el mismo SDG, los programas son Semánticamente Equivalentes, sin embargo, es posible que dos programas Semánticamente Equivalentes no tengan SDG iguales.
4. Una transformación conserva la semántica del programa si, tras efectuarla, cambia los comportamientos computacionales sin alterar el resultado de los cómputos. Una secuencia lo bastante larga de transformaciones que conserven la semántica de un programa puede transformar todos los programas Semánticamente Equivalentes en programas con el mismo SDG.
5. Basado en las reglas 3 y 4, las similitudes semánticas de un programa pueden calcularse comparando el SDG estandarizado del estudiante con los SDG de las soluciones propuestas por el profesor.
6. La similitud semántica del programa es un número real entre 0 y 1. Si el programa obtiene un 1 en similitud semántica, es equivalente al programa propuesto por el profesor y obtiene una nota máxima.
7. Es posible que la similitud semántica del programa sea menor que 1 para todos los modelos propuestos, en ese caso, es posible que el programa entregado por el alumno tenga errores, o bien, sea correcto, pero implemente una solución no prevista por el profesor. En este caso la similitud semántica será la mayor similitud del conjunto.

AutoLEP, a la fecha de 3 de Abril de 2017, no parece encontrarse disponible de forma comercial o gratuita.

Comparación de Pseudocódigo:

Abd Rahman et al (22) proponen una técnica similar a la expuesta por Wang y colegas, si bien, esta aproximación es más simple, pero menos meticulosa.

El sistema implementado hace uso de una lista de palabras claves, similar a un diccionario del lenguaje de programación usado, cada palabra clave está asociada a un *token* dentro del sistema (ejemplo: las palabras *while* y *for* están asociadas al token *loop*). El código sometido a corrección por el estudiante es leído caracter a caracter por el sistema, hasta que este encuentra una coincidencia con alguna palabra clave definida, este proceso se denomina *tokenización*, y genera un pseudocódigo basado en el programa.

Las partes del programa que no están asociadas a un token son consideradas variables o funciones creadas por el estudiante, por medio de un proceso de Estandarización, los nombres de todas las variables y funciones son reemplazados por v_i y f_j respectivamente, donde los subíndices *i* y *j* indican que la variable o función es la *i*-ésima o *j*-ésima encontrada.

El sistema es, entonces, usado sobre un conjunto respuestas modelo proporcionadas por el profesor, realizando una comparación estructural entre los programas del alumno y el profesor, la nota final del trabajo es la similitud máxima entre el programa del alumno y cualquiera de los programas del profesor.

Esta herramienta fue desplegada en los servidores de la *Selangor International Islamic University College Selangor,* para ser probada y aprovechada por sus estudiantes, sin embargo, no ha sido liberada en ninguna forma para su uso por otras instituciones o individuos.

La Tabla 2.1.1 presenta un resumen de las herramientas investigadas y los lenguajes de programación comprendidos por estas.

Nombre	C/C++	Java	Python	Scheme	Haskell	Otros
WEB-CAT	x	x	-	-	-	-
BOSS	x	x	-	-	-	x
Mailing It In	-	x	-	-	-	-
EduComponents	-	-	x	x	x	x
CTPracticals	-	-	-	-	-	x
Fitchfork	x	-	-	-	-	-
Marmoset	x	x	-	-	-	x
ECAutoAssessment Box	-	x	x	x	x	x
JavaBrat	-	x	-	-	-	-
LabRat	-	x	-	-	-	-
AutoLEP*	x	-	-	-	-	-
Pseudocode	x	-	-	-	-	-

Tabla 2.1.1: Software de Corrección Automática y lenguajes comprendidos por estos.

Nombre	Descarga gratuita	Código Abierto	Disponible Para compra	Compatible con Moodle v2.X	Compatible con otra versión de Moodle	Compatible con otro LMS
WEB-CAT	X	X	-	-	-	-
BOSS	-	X	-	-	-	-
Mailing It In	X	-	-	-	-	X
EduComponents	X	X	-	-	-	X
CTPracticals	X	-	-	-	X	-
Fitchfork	-	-	-	-	-	X
Marmoset	X	X	-	-	-	-
ECAutoAssessmentBox	X	-	-	-	-	X
JavaBrat	X	-	-	-	X	-
LabRat	-	-	-	-	-	-
AutoLEP*	-	-	-	-	-	-
Pseudocode	-	-	-	-	-	X

Tabla 2.1.2: Software de Corrección Automática. Disponibilidad y compatibilidad.

2.2 Detección de Plagio y Código Malicioso

Dos de los desafíos más presentes en el ámbito de la corrección automática son la detección de plagio y la protección del computador del corrector. Sin un sistema automático, estas tareas sólo pueden realizarse con la lectura atenta del código entregado por los alumnos, sin embargo, con cursos bordeando los 40 alumnos, el volumen de lectura de código ya comienza a superar la capacidad del instructor de identificar problemas. Por lo anterior, la automatización de estos procesos debe tomarse en consideración al momento de diseñar un sistema de corrección automática. En esta sección se tratará tanto la detección de plagio como la detección de código malicioso, código incorrecto (como pueden ser bucles infinitos) y la protección del medio usado para corregir la tarea.

Ihantola et al (9) sugieren algunas alternativas para controlar este riesgo, entre ellas se encuentran sistemas como Systrace (13), el Módulo de Seguridad de Linux o la Política de Seguridad de Java (23). Otras soluciones implican el uso de expresiones

regulares (24) o el establecimiento de una conexión entre el Cliente (alumno) y el Servidor (corrector) que permita correr el programa en el lado del cliente (12).

Butakov et al (25) proponen un sistema computarizado para la detección del plagio digital. Este sistema convierte los archivos entregados a texto plano, para luego usar el algoritmo de *Winnowing* (26), el cual busca similitudes entre los textos. El sistema propuesto permite la búsqueda de plagios en escala local, tanto como global.

Pieterse (2) define un límite de eficiencia, es decir, un tiempo límite tras el cual el proceso que corre un programa entregado se mata, esta capacidad es bastante útil a la hora de salir de un bucle infinito dentro de un programa mal codificado. Para la detección de plagio, se escribió un script que calcula el hash MD5[13] de cada archivo, el cual debería ser idéntico para dos archivos idénticos, lo cual permite identificar un archivo copiado, desgraciadamente, si los archivos no son idénticos, el has MD5 tampoco lo es, por lo cual algo tan sencillo como cambiar un comentario es suficiente para engañar este método. De interés resulta acotar que, según Pieterse, muchos alumnos ni siquiera realizan un esfuerzo nominal por disfrazar su copia (2).

Otro posible método, si bien, que requiere algo de trabajo del corrector, es usar un sistema de corrección estática, tales como los enunciados por los equipos de Wang (20) y Abd Rahman (22), no sólo como una forma de comparar los programas con las respuestas modelos, sino también comparando los programas de los alumnos entre ellos mismos, para detectar plagio, o compararlos con programas atacantes modelo, con el fin de detectar programas similares.

2.2.1 Software Especializado:

[13] https://es.wikipedia.org/wiki/MD5

Como se da el caso con todas las otras áreas de la pedagogía, existen programas especializados en la detección de plagio en tareas de programación.

Sherlock: (27)

El Software Sherlock, desarrollado en la Universidad de Warwick, Inglaterra, parte del supuesto de que un alumno que comete plagio del trabajo de otro alumno, lo hace porque no tiene claro cómo el programa resuelve el problema en cuestión, por lo tanto, sus intentos por disfrazar el plagio serán lo menos disruptivos posibles con el código que sí funciona, de esta forma, los algoritmos necesarios para detectar un programa plagiado no necesitan ser de lo más sofisticado. Sherlock es utilizado como un complemento de la herramienta BOSS de entrega de tareas (27) (8), como tal, comprende los mismos lenguajes que esta, indicados más arriba en la tabla 2.1.1.

Sherlock utiliza una técnica de comparación estructural incremental, la cual consta de dividir un programa en *tokens* indivisibles, similar a la idea de Wang et al (21), expuesta anteriormente. Con esto, se puede establecer una similitud entre dos programas, incluso si ciertos cambios han sido realizados (sentencias *if* anidadas reemplazadas por *case,* ciclos *for* por ciclos *while*, etcétera). Los programas se comparan cinco veces:

- En su forma original.
- Tras remover todos los espacios blancos posibles.
- Tras remover todos los comentarios.
- Tras remover todos los espacios blancos y comentarios.
- Tras terminado el procesos de tokenización.

Plague: (28)

Desarrollado por G. Whale, de la *University of New South Wales*, Australia, Plague divide la detección de plagio en dos etapas: Una etapa de Filtrado, de baja selectividad y una etapa de Asociación, de alta selectividad.

Durante la primera etapa se representa la estructura del programa mediante una notación similar a una expresión regular, una segunda transformación convierte esta expresión regular en una secuencia ordenada de términos cuantitativos, esta secuencia se denomina *perfil de estructura.*

La segunda etapa crea un grafo no dirigido de similitud de programas, usando todos los programas de la misma entrega para el mismo curso como conjunto, basado en los perfiles de estructura, en el cual, la longitud de cada arco del grafo representa la similitud estructural entre ambos programas, en esta representación, un arco largo entre dos programas representa una similitud vaga, que bien podría tratarse de una coincidencia, mientras que un arco corto indica una similitud sospechosa y, por lo tanto, posible plagio.

Plague comprende tareas en lenguaje Pascal y fue diseñado como evidencia de que el entonces popular método de conteo de atributos para detección de plagio era deficiente, como es software diseñado en los años 90, no se encuentra disponible comercialmente.

<u>YAP:</u> (29)

El sistema YAP, desarrollado por Michael Wise, de la Universidad de Sydney, Australia, opera de manera similar a Plague, sin embargo, utiliza un proceso de *tokenización* más sencillo que el de perfilamiento estructural de Plague, optimizando con la portabilidad y velocidad en mente. YAP entrega resultados basados en un porcentaje de *tokens* similares en ambos archivos, lo cual hace más fácil el juzgar si dos archivos similares se deben a una coincidencia o a un plagio.

YAP fue comparado con Plague a modo de prueba de su efectividad, tras lo cual se concluyó que ambos sistemas son aproximadamente igual de efectivos. Desgraciadamente, no se encuentra disponible para uso hoy en día.

2.3 Desarrollo sobre Moodle.

Moodle (30) es un sistema *Open Source* de manejo y gestión de cursos, desarrollado con cursos universitarios en mente, proveído de forma gratuita desde su sitio web, bajo la Licencia Pública de GNU.

Una instalación típica de Moodle contiene tres elementos (14):

1. Un directorio para los archivos PHP que contienen el código fuente de la aplicación.
2. Un directorio con archivos de datos sobre los cursos y usuarios,
3. Una Base de Datos que define los diferentes elementos que componen el sistema.

La unidad básica de organización de Moodle es el *Curso*, al cual se accede a través de una página web. El curso es dividido en secciones, las cuales pueden representar temas, semanas, clases o alguna otra división que tenga sentido, en cada una de estas secciones es posible incluir recursos o actividades que complementen el aprendizaje, estas últimas pueden asignarse como tareas, ya sea para realizar en clases o en casa.

Un elemento importante de la interfaz de Moodle son los *bloques*, los cuales pueden verse a los lados de la página de cualquier curso, los bloques proveen atajos y herramientas de control y gestión del perfil del usuario.

Moodle soporta tres tipos de usuario diferentes: *Administrador*, *Profesor* y *Estudiante*, diferenciados por los privilegios que estos tienen una vez inscritos en un curso.

Cada actividad de Moodle corresponde con un *Módulo.* Instalar un nuevo módulo en Moodle es una tarea sencilla, que sólo requiere que el directorio con el código fuente de la nueva actividad tenga el mismo nombre que ésta. Moodle reconoce automáticamente cuando un nuevo módulo es añadido al directorio de código fuente. La instalación de *bloques* es similar: Una copia del directorio de cualquier nuevo bloque debe ir en el directorio de código fuente. Moodle provee varios módulos básicos, tales como foros, quizes, wikis, encuestas, páginas web, hipervínculos y tareas entre otros, al

igual que una gran variedad de bloques predefinidos como calendario, actividad reciente y administración de cursos. Esta modularidad y buena definición de interfaces permite la creación de nuevas actividades (módulos) y bloques sin la necesidad de modificar el núcleo de Moodle.

2.4 Arquitectura

Como se mencionó previamente, el Departamento de Informática de la Universidad Técnica Federico Santa María proporciona a sus alumnos con un repositorio Git[14] asociado con su cuenta de correo del propio departamento, razón por la cual, resulta de interés examinar el sistema Marmoset (15) por Spacco et al, el cual utiliza un repositorio de código para mantener un registro histórico de los progresos de cada estudiante en el desarrollo de cada tarea. El sistema Marmoset cuenta con una arquitectura relativamente simple, la cual consta de un servidor de Entrega de tareas y uno de Construcción y Pruebas. El servidor de entregas actúa como un intermediario entre el servidor de pruebas y la estación de trabajo del alumno, quien podrá hacer entrega de su trabajo por medio de línea de comandos o de un plugin para IDE como Eclipse.

Por supuesto, la Universidad cuenta también con un servidor de Moodle, y usa este LMS como su forma primaria de comunicación entre el profesor y los alumnos de la mayoría de los cursos del Departamento de Informática, sabiendo esto, es de particular interés el caso de CTPracticals.

CTPracticals (14) hace uso de una extensión de Moodle para administrar la entrega y revisión de tareas. Para esto, la herramienta define ciertas tablas nuevas en la base de datos usada por Moodle, las cuales extienden sus dos tablas nativas *Courses* (Cursos) y *Users* (Usuarios), estas tablas incluyen *Practical* (la tarea), *Team* (los autores), *Work* (la entrega), *Tester* (la configuración de la prueba) y *Work State* (El estado de la entrega). El motor de autocorrección

[14] https://git-scm.com/

de CTPracticals es externo al módulo en sí. Como una extensión de Moodle, CTPracticals hace uso de la misma arquitectura utilizada por el servidor de Moodle de la institución que lo usa.

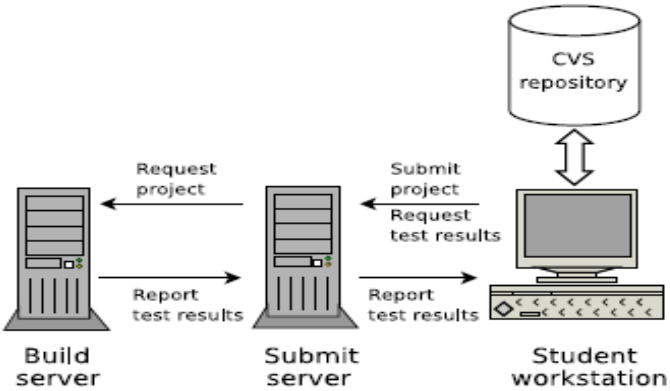


Figure 1: Architecture of MARMOSET

Figura 2.4.1: Arquitectura de Marmoset, fuente: Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses

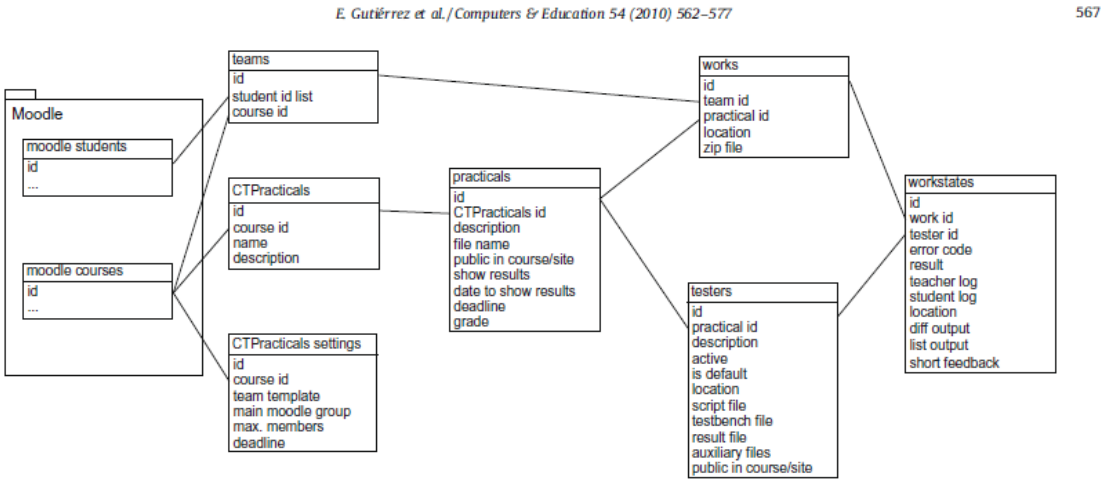


Fig. 3. Tables added by *CTPracticals*.

Figura 2.4.2: Tablas añadidas a Moodle por el módulo CTPracticals. Fuente: A new Moodle module supporting automatic verification of VHDL-based assignments. Gutiérrez et al. 2010

Una arquitectura similar es la usada por Jelemenská y Cicák (31) para el módulo que ellos desarrollaron e implementaron en la *Slovak University of Technology Bratislava*, en Eslovaquia. El módulo extiende las tablas de usuario y curso para agregar funcionalidades como entregas de borrador, las cuales, probando su utilidad, han sido implementadas en el núcleo de Moodle para el tiempo en que este documento se ha escrito.

Una alternativa más compleja, pero probablemente más segura y robusta, es la utilizada por BOSS (8), una arquitectura cliente-servidor en la cual tres servidores distintos son utilizados, un servidor para Estudiantes, uno para Staff, y uno para las Pruebas.

El servidor de estudiantes se usa para la recepción de entregas de tareas autentificación del trabajo y detección de plagio. El servidor de estudiantes puede comunicarse con el de pruebas en caso de que se le permita al estudiante realizar entregas de borrador antes de la entrega final, con el fin de poder probar su código y obtener algo de retroalimentación antes de la fecha límite de la entrega.

El servidor de staff sólo puede ser usado por miembros autorizados y certificados del staff de enseñanza del curso, vale decir, profesores, ayudantes y coordinadores. Todo el proceso de corrección, evaluación y administración de las notas se realiza en este servidor. El servidor de staff puede comunicarse con el servidor de pruebas por razones obvias.

El servidor de pruebas realiza la tarea por la que recibe su nombre en uno de dos modos posibles: por entrega y por curso. Una corrección por entrega es realizada cuando el alumno entrega su trabajo a través del servidor de estudiantes, esperando recibir retroalimentación sobre su trabajo, el servidor realiza este tipo de corrección para cualquier tarea que reciba antes de la fecha límite. Una corrección por curso se realiza después de la fecha límite sobre todas las entregas recibidas, esta corrección no entrega retroalimentación y el resultado de esta son las calificaciones del curso.

Los Servidores de BOSS operan sin privilegios de súper-usuario, para que, si alguno fuera atacado, el daño pueda controlarse. Como la mayoría de las

funcionalidades de BOSS se ubican en los servidores, los clientes se mantienen relativamente ligeros y seguros.

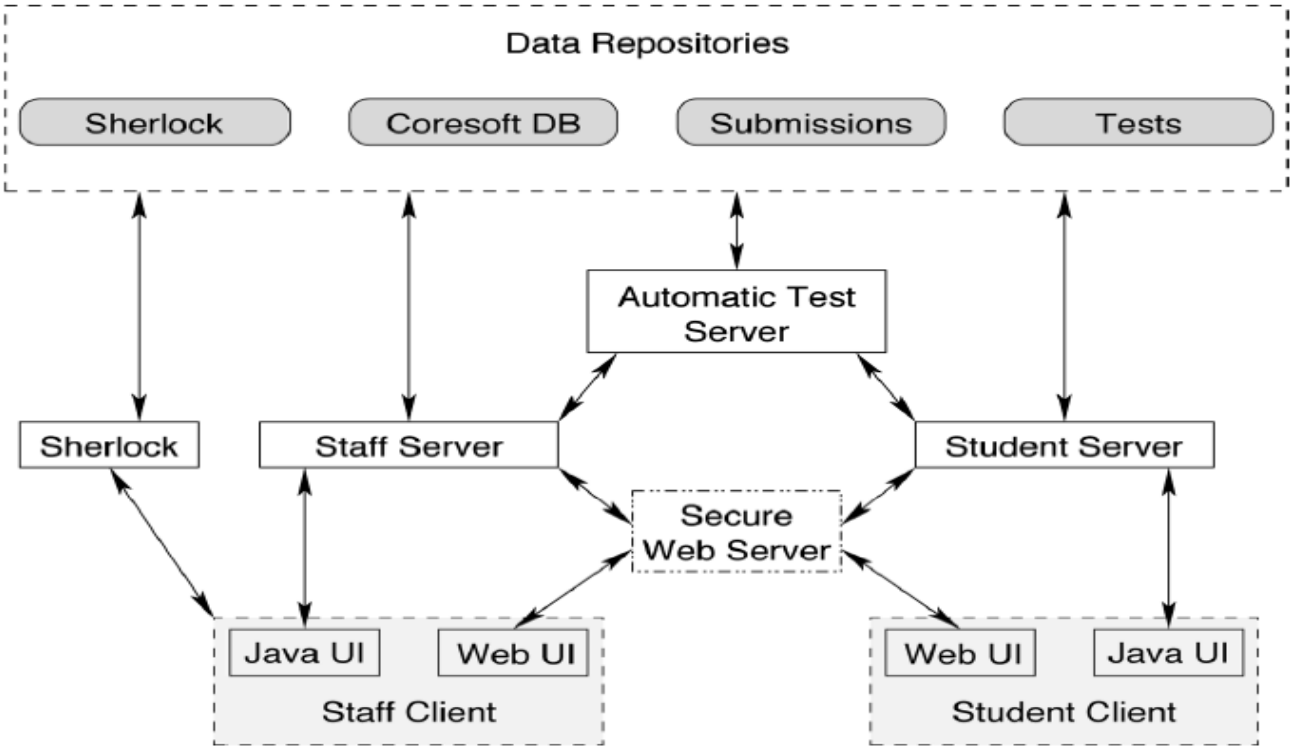


Fig. 5. The BOSS architecture.

Figura 2.4.3: Arquitectura de BOSS. Fuente: The BOSS Online Submission and Assessment System

Capítulo 3: Solución Implementada:

Habiendo considerado tanto el trabajo realizado por otros investigadores y desarrolladores, así como la realidad actual del Departamento de Informática de la Universidad Técnica Federico Santa María, con miras hacia establecer los cimientos de trabajos futuros, y tomando en cuenta los objetivos planteados en este mismo documento, se ha optado por el desarrollo de una aplicación capaz de corregir automáticamente una lista de tareas de programación, además de construir una matriz de plagio que indica el porcentaje de similitud entre todos los posibles pares de tareas entregadas.

Para el segundo semestre de 2016, y desde tan temprano como el curso de Lenguajes de Programación, en el cuarto semestre de la malla vigente (ver figura 1.1.1), se insta a los alumnos a utilizar un repositorio Git para la entrega de tareas (32), a diferencia de simplemente entregar las tareas vía Moodle. Tomando en cuenta lo anterior, así como la investigación realizada, se concluye que es viable implementar el sistema de autocorrección como una extensión de Moodle, sin embargo, como Moodle no siempre se usa, sería recomendable también poner al alcance de profesores y ayudantes el ejecutable de la aplicación desarrollada.

Aunque el uso de otros sistemas ya establecidos fue considerado, se descubrió durante la investigación previa al presente trabajo, que no existe literatura sobre el tema desarrollada por investigadores nacionales. El desarrollo de la herramienta descrita a continuación tiene por objetivo construir las bases de varios otros trabajos posibles a futuro, permitiendo a investigadores por venir la posibilidad de expandir y comparar este sistema con otros establecidos e incentivar nuevos temas de investigación desde el punto de vista técnico y, de ser posible en el mediano a largo plazo, desde el punto de vista del aprendizaje.

En el presente capítulo se detalla la aplicación desarrollada, denominada de ahora en adelante SAC (Sistema Automático de Corrección), presentando diagramas de flujo para explicar su funcionamiento.

3.1 Sistema Automático de Corrección (SAC):

SAC es una aplicación programada en el lenguaje Java, la cual es capaz de automatizar la corrección de una lista de tareas y detectar posibles intentos de plagio entre estas. Los datos de salida entregados por SAC consisten en archivos .csv (*coma separated values*), los cuales pueden leerse fácilmente usando Microsoft Excel, Open Office Calc, Google Spreadsheets o similares alternativas. SAC genera dos archivos: uno que contiene los resultados de la corrección automática para cada alumno, para cada prueba y observaciones especiales (ausencia de archivo *makefile*, falta de comentarios y/o indentación, no genera ejecutable, etc.), el segundo archivo es una matriz de plagio, la cual contiene el porcentaje de similitud entre cada par de tareas.

SAC requiere de un archivo de parámetros, el cual debe ser un archivo de texto plano con los siguientes valores y en el siguiente orden:

1. El lenguaje que se espera corregir. Este parámetro debe ser “C” o “Python”.
2. El modo de corrección. Este parámetro debe ser “Completo” o “Plagio”. Qué significan estos valores será explicado más adelante en este mismo capítulo.
3. La ruta del directorio en el cual las tareas se encuentran, relativa al ejecutable de SAC. Las tareas deben estar descomprimidas.
4. La ruta del directorio en el cual se encuentran los datos de entrada de prueba, relativa al ejecutable de SAC.
5. La ruta del directorio donde se encuentran los datos de salida esperados, relativa al ejecutable de SAC.

6. La ruta del directorio para los archivos de código estandarizado, relativa al ejecutable de SAC. Necesario para ambos modos.

Adicionalmente, SAC requiere dos archivos de texto plano llamados Diccionario C y Diccionario Python, estos archivos han sido elaborados y provistos por el propio desarrollador.

3.1.1 El Proceso de Corrección:

El proceso de corrección de SAC consta de tres subprocesos:

1. Corrección dinámica.
2. Estandarización de código.
3. Detección de plagio.

En modo Completo, SAC lleva a cabo estos tres procesos, mientras que en el modo de Plagio, sólo realiza los últimos dos. La razón para desarrollar dos modos de operación distintos es simple: no todas las tareas reciben datos de entrada de forma estándar, por lo cual, un modo de detección de plagio hace que SAC sea útil también en el caso de tareas que reciben datos de entrada del usuario vía teclado, las cuales, como se mencionó en el capítulo 1, no podrían corregirse automáticamente. El Diagrama 1 muestra el proceso general de operación de SAC, los procesos específicos serán detallados a continuación.

Corrección Dinámica:

Wang et al (21) se refieren a la corrección dinámica como aquella en la que se ejecuta el código. Como no todas las tareas asignadas por el departamento tienen necesariamente que depender de una entrada y salida establecida, este proceso se deja como opcional.

El proceso de Corrección dinámica empieza buscando el código en la carpeta de la entrega realizada por el estudiante, de no encontrarse, se escribirá una observación de “No hay código” en el archivo de resultados.

En el caso de que el lenguaje especificado para la corrección sea C, SAC buscará un archivo con nombre Makefile en la carpeta y, similar al caso anterior, se escribirá una observación “No hay makefile” en el evento de no encontrarse.

Si el código o el makefile no fueran encontrados, el proceso de corrección dinámica se detiene y se pasa a estandarizar el código (en caso de makefile faltante, pero código presente) o a la siguiente tarea (en caso que no haya código para estandarizar).

Habiendo encontrado el código y, de ser necesario, el archivo makefile requerido, la tarea es compilada y ejecutada. Para la ejecución, se copia una entrada de la carpeta correspondiente entregada como parte de los argumentos y se le deposita en la carpeta de la tarea, para luego renombrarlo como input.txt.

Posteriormente, la salida generado por el programa (el cual debe llamarse output.txt) es comparado línea por línea con los datos de salida esperado según las pruebas, un puntaje es asignado para cada prueba, el cual es calculado de la siguiente forma:

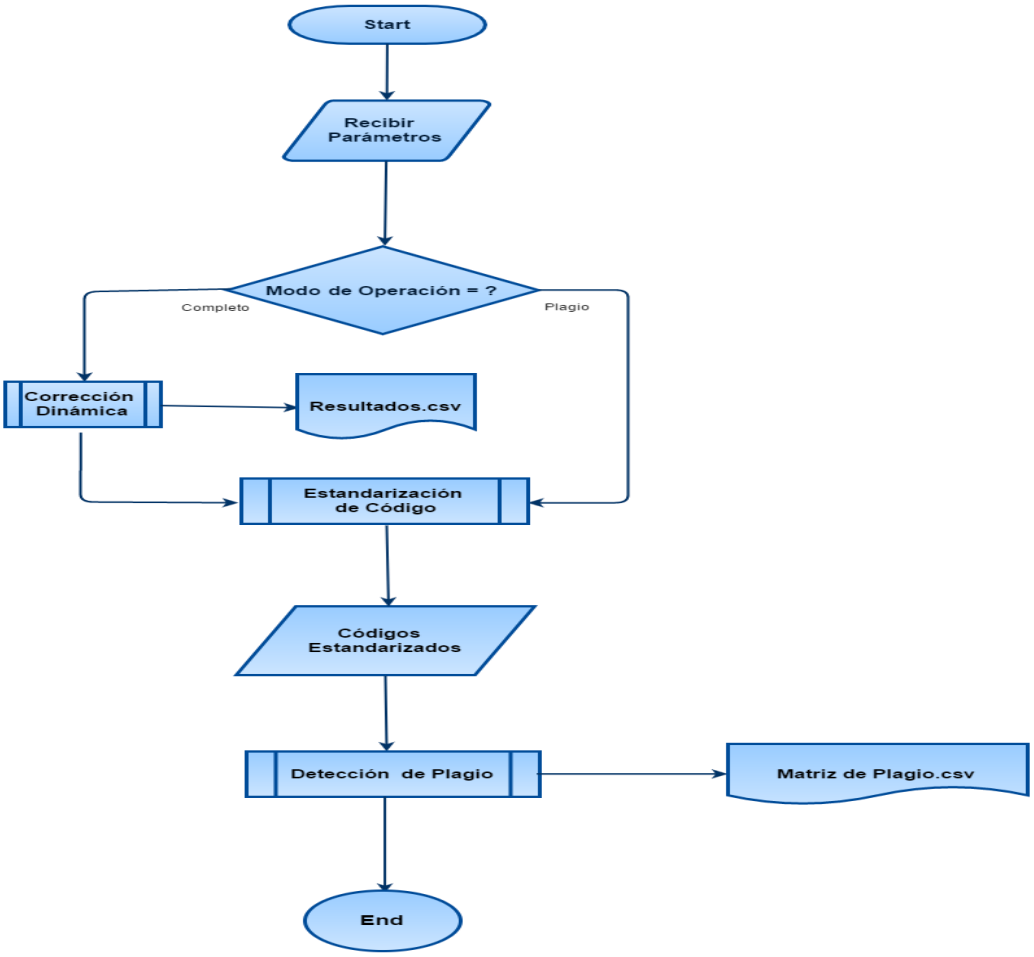


Figura 3.1.1.1: Visión General de SAC. Fuente: Elaboración Propia.

$$P_i = 100 * \frac{Ci}{Ti}$$

Ecuación 1: Porcentaje de Similitud entre archivos

En donde:

- P_i es el porcentaje de similitud entre la salida generada por el alumno y la *i*-ésima prueba.

- C_i es la cantidad de líneas que coinciden entre la salida generada por el alumno y el esperado según la *i*-ésima prueba.
- T_i es el número total de líneas de salida de la *i*-ésima prueba.

La comparación de líneas se realiza usando el método *equals()*[15] de Java, el cual retorna verdadero para dos *strings* iguales y falso para dos *strings* diferentes, como el método usado para leer los archivos de salida generados por los alumnos es *readLine()*[16], la comparación entre strings debería retornar resultados confiables.

Una vez que las pruebas han concluido, se procede a iniciar el proceso de estandarización de código. El Diagrama 2 representa el proceso de Corrección Dinámica para una tarea, este proceso se repite para todas las tareas encontradas.

[15] https://www.tutorialspoint.com/java/java_string_equals.htm
[16] https://www.tutorialspoint.com/java/io/bufferedreader_readline.htm

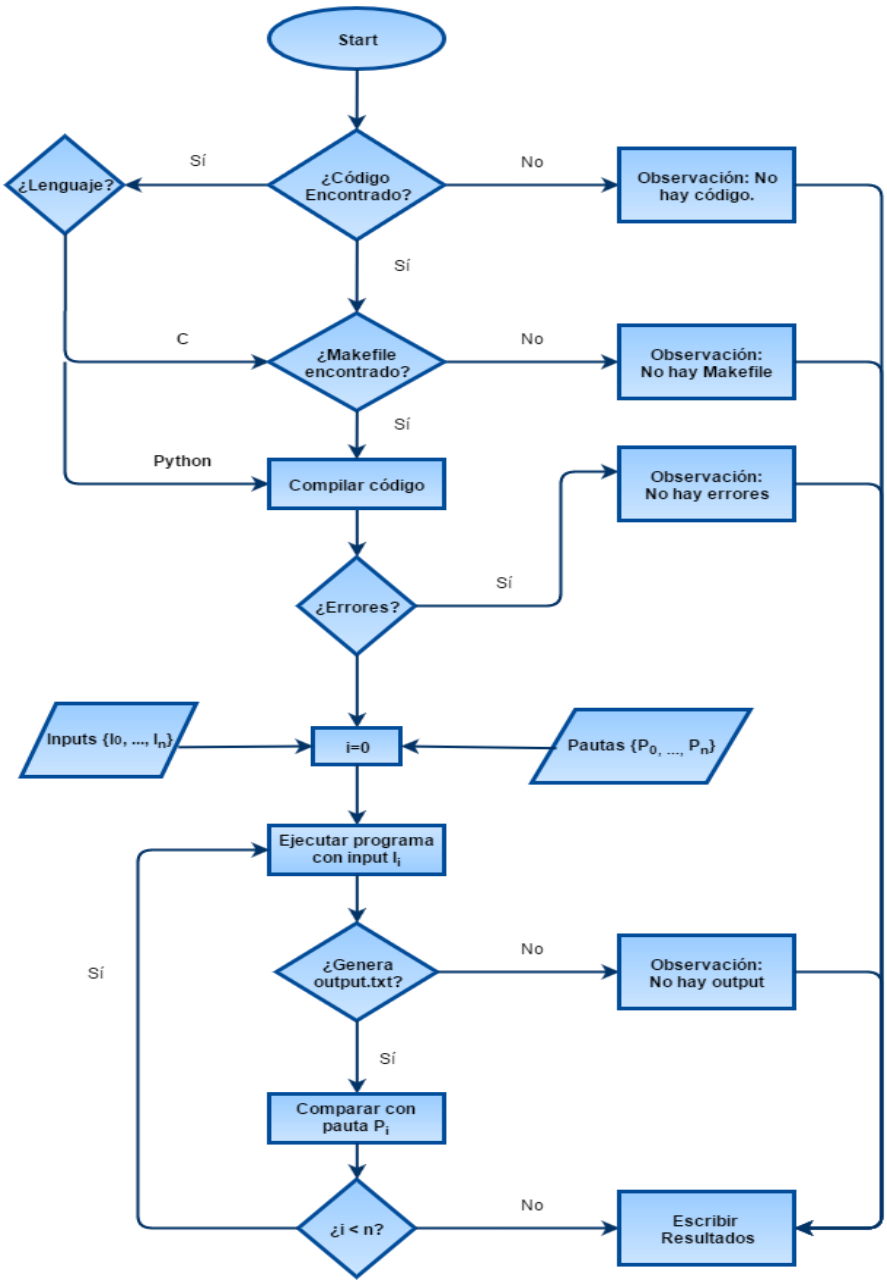


Figura 3.1.1.2: Proceso de Corrección Dinámica. Fuente: Elaboración Propia.

Estandarización de Código:

El proceso de estandarización es similar al proceso de *tokenización* expuesto por Abd Rahman y su equipo (22), aunque la presente implementación es un poco más sencilla.

El código fuente se lee línea por línea. Una vez se lee una línea del archivo, se usan los métodos de Expresiones Regulares de Java para dividir la línea en palabras, buscando cada una de estas palabras en un diccionario del lenguaje de programación usado. El diccionario a usar depende del parámetro entregado al principio del proceso, por lo que es importante que los diccionarios se mantengan en la misma carpeta que el ejecutable de SAC. Si alguna de las palabras no se encuentra en el diccionario, se procede a encontrar la palabra en su contexto original, usando expresiones regulares, se determina si la palabra en cuestión corresponde a variable o a una función. Para un ejemplo de esto, véase el siguiente código:

```
void main (){
      int numero;
      numero = 0;
      //foo(a,b) = a^b
      numero = foo(5,4);
      printf (“%d”, numero);
}
```

SAC lee línea por línea y analiza cada palabra de cada línea. Las palabras void y main de la primera línea están incluidas en el diccionario del lenguaje, por lo que SAC no opera en ellas. En la segunda línea, sin embargo, encuentra la palabra numero, que, como podemos ver, es el nombre de una variable. Usando una expresión regular, SAC

determina que numero no es una función, por lo que debe ser el nombre de una variable, de la misma forma, determina que foo es una función, que “%d” es un *string* y que el comentario es justamente eso. SAC conserva una lista de las variables, funciones y estructuras definidas por el alumno. Cuando SAC identifica una palabra indefinida por su naturaleza (variable, función, estructura, *string*, o comentario), procede a buscar en los registros del programa, si la palabra no está registrada y corresponde con una variable, función o estructura, se le agrega a la lista correspondiente, para luego reemplazar la palabra en la línea por v_i , f_j , o s_k , denotando que esta es la *i*-ésima variable, *j*-ésima función o *k*-ésima estructura, los *strings* se reemplazan por la palabra “mensaje” y los comentarios por la palabra “comentario”. Continuando con el ejemplo anterior, supóngase que la variable numero y la función foo son la primera variable y función encontradas respectivamente. Luego de estandarizar las líneas, estas se escribirían en un archivo llamado pseudocodigo-XXX.txt, donde XXX es el nombre, rol o alguna otra identificación del estudiante, obtenida del nombre del directorio donde se encuentra la tarea. El archivo contendría el siguiente código estandarizado:

```
void main (){
      int v1;
      v1 = 0;
      //comentario
      v1 = f1(5,4);
      printf (mensaje, v1);
}
```

Los archivos de código ya estandarizado son trasladados al directorio reservado para ellos, el cual corresponde con la sexta línea del archivo de parámetros. El diagrama número 3 detalla el proceso para una tarea determinada, este proceso, una vez más, se realiza sobre todas las tareas entregadas.

Detección de Plagio:

El proceso final de la operación de SAC es el de detectar el posible plagio entre tareas de los estudiantes. Este proceso requiere que todos los archivos de código estandarizado se encuentren en un mismo directorio.

SAC comienza creando una matriz bidimensional de dimensiones *nxn*, donde *n* es el número de archivos en el directorio. Esta es la matriz de plagio, cada posición de la cual registrará el porcentaje de similitud entre dos tareas. En un inicio, cada posición de la matriz se inicializa con valor igual a -1, excepto por los valores de la diagonal, los cuales se inician en 100, ya que una tarea es 100% similar a sí misma.

SAC recorre la lista de archivos de código encontrados en el directorio, la cual se guarda como un archivo de objetos clase File[17], para dos tareas dadas, con los índices *i y j*, SAC revisa el valor de la posición (*i,j*) en la Matriz de Plagio, si el valor de la casilla es igual a -1, entonces ambas tareas son comparadas.

La comparación comienza recolectando cada función de un determinado archivo en una lista de strings, estas listas se reúnen, a su vez, en una lista, tal como lo indica la figura 3.1.1.4.

Sean, entonces, dos tareas, *A* y *B*, con funciones $\{a_1, a_2, a_3,...,a_n\}$ y $\{b_1, b_2, b_3,...,b_m\}$ respectivamente, cada función tiene un número de líneas de código $\{la_1, la_2, la_3,...,la_n\}$ y $\{lb_1, lb_2, lb_3,...,lb_m\}$. La similitud entre estas dos tareas se denominará como **Sim (A, B).**

Un lector atento notará que al sumar la longitud de cada función, se puede obtener el largo total de la tarea, medido en líneas de código, de esta forma, llamando a estas longitudes *LA* y *LB* respectivamente:

[17] https://docs.oracle.com/javase/7/docs/api/java/io/File.html

$$\boldsymbol{LA} = \sum_{i=1}^{n} \boldsymbol{la_i}$$

$$\mathbf{LB} = \sum_{i=1}^{m} \mathbf{lb_i}$$

Ecuación 2: Longitud total de dos programas A y B en líneas de código.

El proceso comienza tomando la función a_1 y comparándola con todas las funciones de la tarea B, una por una. De cada una de estas comparaciones, se obtiene un porcentaje de similitud entre la función a_1 y una función b_i de la tarea B, este porcentaje se denominará **Sim (a_1, b_i)** y se obtendrá de la manera más directa:

$$\boldsymbol{Sim}\,(\boldsymbol{a_1}, \boldsymbol{b_i}) = \mathbf{100} * \boldsymbol{n}/\boldsymbol{la_1}$$

Ecuación 3: Porcentaje de Similitud entre una función a_1 y una función b_i

Donde n corresponde al número de líneas que coinciden entre a_1 y b_i.

Una vez se obtienen los *m* porcentajes de similitud, se guardan en un arreglo, el cual representa la *similitud de* ***a₁*** *con respecto a* ***B*** (en adelante, denominado ***Sa₁B***).

Una vez se tiene el arreglo Sa_1B, se selecciona el mayor elemento de este, la mayor similitud entre la función a_1 con la tarea *B,* y se traslada ese valor a un segundo arreglo, el cual representa la *similitud de la tarea* ***A*** *con respecto a la tarea* ***B*** (en adelante, denominado como ***SAB***).

El proceso anterior se repite para todas las funciones de la tarea *A.* Terminado esto, el arreglo *SAB* estará lleno, con cada posición conteniendo un número entre 0 y 100, cada cual representa la mayor similitud entre una función a_i de la tarea *A* con respecto a la totalidad de la tarea *B.*

A continuación se determina qué fracción de las líneas de código de la tarea A pertenecen a cada función, esta fracción fAa_i, donde 0<i<(n+1), se determina dividiendo la longitud de la función la_i por la longitud de la tarea *LA.*

$$\mathbf{fAa_i} = \mathbf{la_i}/\mathbf{LA}$$

Ecuación 4: fracción del código de la tarea A que corresponde a la función a_i.

Cada fracción fAa_i se guarda en un arreglo ***fA***. La similitud total entre las tareas A y B se determina multiplicando cada elemento del arreglo *SAB* con su homólogo del arreglo *fA* y sumando todos los productos resultantes.

$$\mathbf{Sim}\ (\mathbf{A}, \mathbf{B}) = \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{fAa_i} * \mathbf{SAB_i}$$

Ecuación 5: Porcentaje de Similitud entre dos tareas A y B

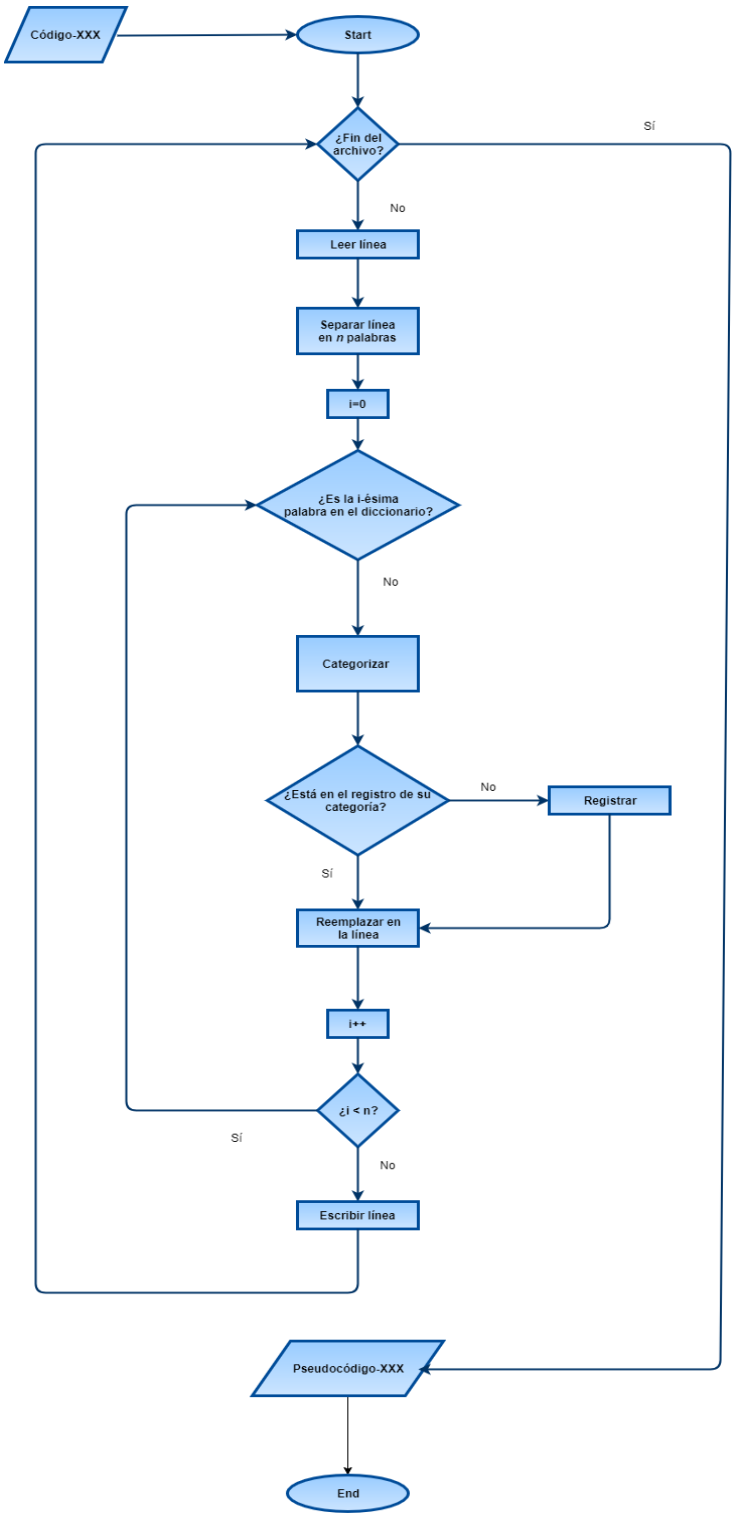


Figura 3.1.1.3: Proceso de Estandarización de Código. Fuente: Elaboración Propia.

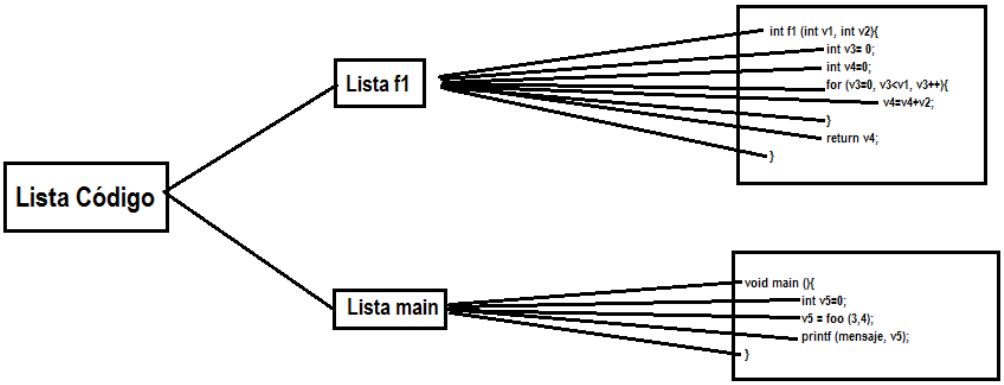


Figura 3.1.1.4: Estructura usada por SAC para comparación y detección de plagio. Elaboración Propia.

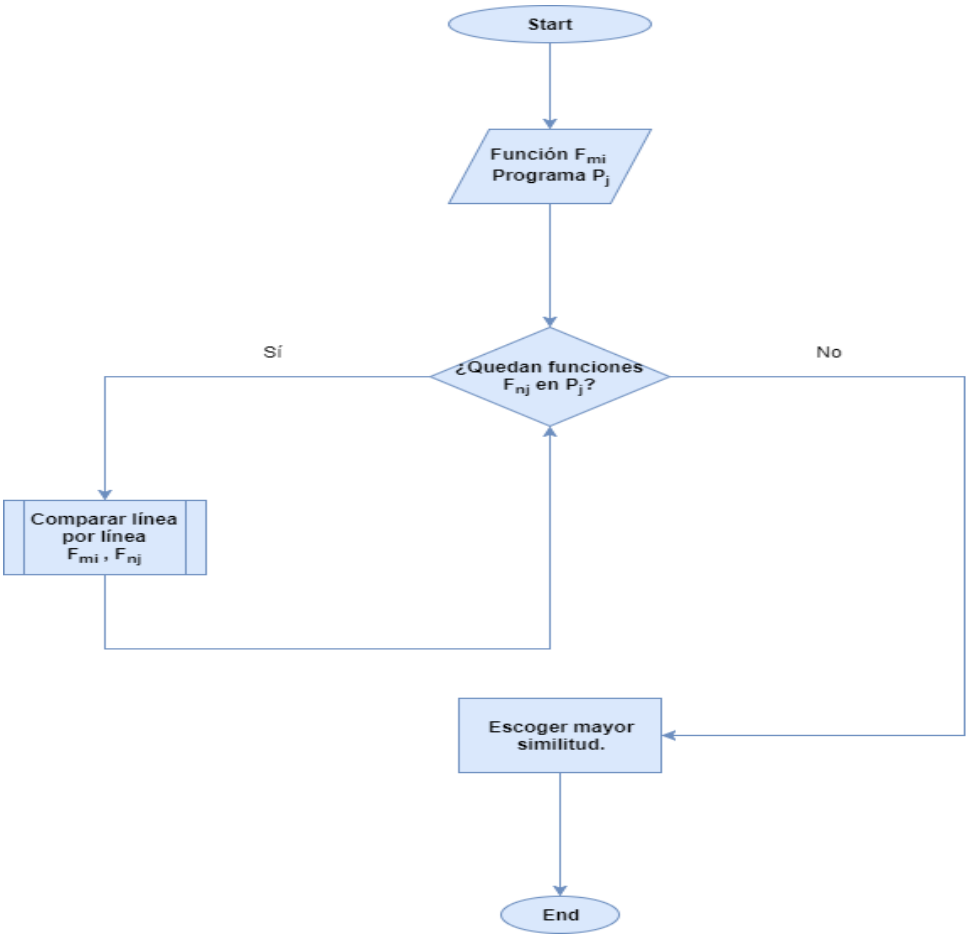


Figura 3.1.1.5: Comparación entre funciones. Fuente: Elaboración Propia.

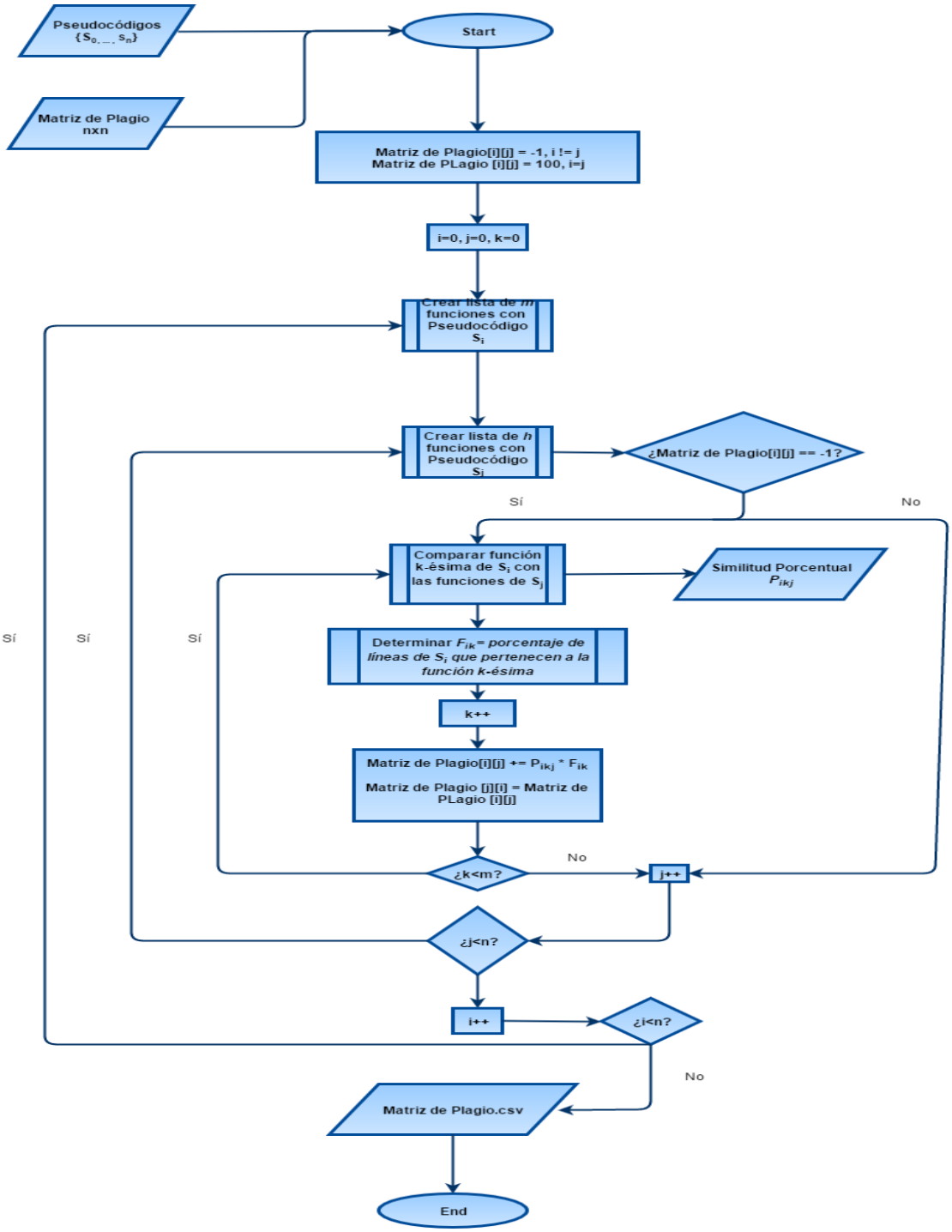

Pseudocódigos
{S0,..., Sn}
Matriz de Plagio
nxn
Start
Matriz de Plagio[i][j] = -1, i != j
Matriz de PLagio [i][j] = 100, i=j
i=0, j=0, k=0
Crear lista de m funciones con Pseudocódigo Si
Crear lista de h funciones con Pseudocódigo Sj
¿Matriz de Plagio[i][j] == -1?
Sí
No
Comparar función k-ésima de Si con las funciones de Sj
Similitud Porcentual Pikj
Determinar Fik = porcentaje de líneas de Si que pertenecen a la función k-ésima
k++
Matriz de Plagio[i][j] += Pikj * Fik
Matriz de Plagio [j][i] = Matriz de PLagio [i][j]
¿k<m?
No
j++
¿j<n?
i++
¿i<n?
No
Sí
Sí
Sí
Matriz de Plagio.csv
End

Figura 3.1.1.6: Detección de Plagio. Fuente: Elaboración Propia.

3.1.2 SAC: No es una Extensión de Moodle:

SAC se ha desarrollado como una aplicación independiente, no como extensión de Moodle y, aunque podría desarrollarse un plugin que lo implemente como una herramienta de corrección, existen argumentos para no hacerlo.

El profesor Sven von Brand, quien facilitó información y tareas de alumnos para realizar las pruebas de SAC, dijo que las tareas de su asignatura se entregan por medio de un repositorio Git, provisto a los alumnos por el Departamento de Informática, esta modalidad de entrega de tareas anima a los alumnos a, desde temprano en la carrera, aprender a utilizar repositorios de código como control de versiones y para mantener su trabajo en un ambiente seguro en caso de robo de computador u otra calamidad. En el ambiente laboral, para el cual la universidad prepara a sus alumnos, esta habilidad es cada vez más vital y el integrar una extensión de Moodle para facilitar la corrección animaría a profesores y ayudantes a volver a la modalidad anterior de entrega de tareas. Bajo este contexto, SAC se ha diseñado como un programa autónomo, el cual no requiere de Moodle para funcionar y permite a los educadores infundir en sus alumnos habilidades que serán más útiles en su vida profesional.

La seguridad es otra arista importante del asunto, al ser una aplicación autónoma, SAC debe ejecutarse en el computador del corrector y no en el servidor de Moodle de la Universidad ni en servidores destinados exclusivamente para ello, de esta forma, un programa malicioso podría dañar solamente un sistema personal. Aunque este escenario no es ideal y es perfectamente evitable si el corrector ejecuta SAC como un usuario sin privilegios, en el peor de los casos, el daño no lo recibiría un servidor de la Universidad, lo cual afectaría no solo al profesor y al curso en el cual el atacante estudia, sino potencialmente a todos los estudiantes y profesores del DI.

Por las razones previamente expuestas y en concordancia con los objetivos planteados en el capítulo 1 del presente documento, se concluye que una extensión de Moodle es factible y posible, pero de momento, no recomendable.

3.1.3 Detección de Plagio: ¿Por qué SAC y no otro software?

El lector observador probablemente se esté preguntando qué beneficios ofrece SAC, un sistema experimental, sobre Sherlock, Yap o Plague que están probados, la verdad es que no hay una razón real para usar uno por sobre el otro, pero el mayor beneficio podría cosecharse de usar SAC en conjunto con otro sistema de detección de plagio, ya que dos métodos diferentes de búsqueda podrían detectar casos de fraude que el otro sistema no detecta, así como ocurrió con las pruebas de Plague (28), en las que al usar varios sistemas sobre la misma muestra, se revelaron casos de plagio en Plague que otros sistemas fallaron en hallar.

Capítulo 4: Validación de SAC:

Para la validación de SAC, se realizaron dos pruebas diferentes, una prueba básica con programas rudimentarios, la cual estaba orientada a medir los tiempos de operación del programa en modo completo y una prueba con tareas reales, orientada a probar la función de detección de plagio. Durante esta sección se detallarán los errores encontrados durante las pruebas, con el fin de explicar las diferencias en el tiempo de ejecución y los resultados obtenidos para la misma muestra de prueba.

Ambas pruebas se realizaron en el mismo ambiente, una laptop Toshiba Satellite C845-SP4221SL, cuyas características se detallan a continuación:

Tabla 4.1: Especificaciones del sistema utilizado para las pruebas.

Procesador	Intel Pentium B950 2.1 GHz
Memoria RAM	4 GB
Sistema Operativo	Ubuntu 14.04.5
Disco Duro	500 GB

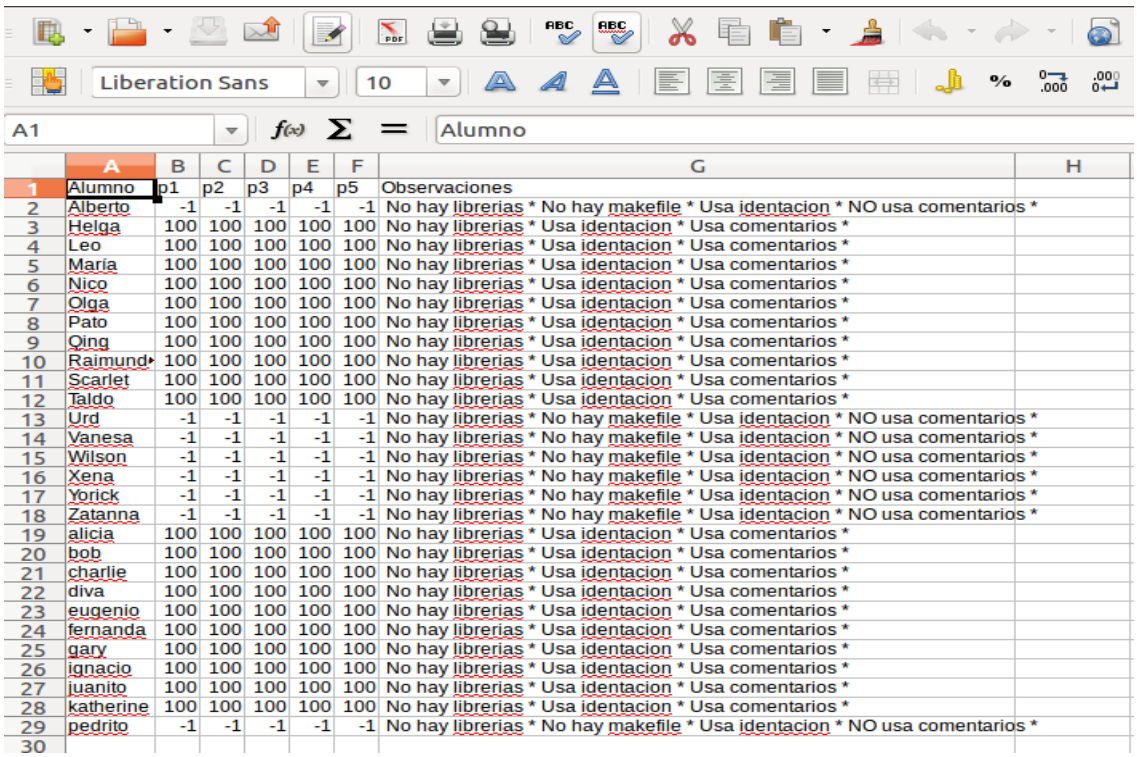
4.1 Prueba Básica:

Para esta prueba se utilizaron programas sencillos, de elaboración propia, los cuales reciben datos de entrada desde un archivo de texto llamado input.txt, se realizaron en total dos muestras de 28 tareas cada una, una en lenguaje C y una en python, se crearon también 5 archivos de entrada y 5 archivos de pauta conteniendo la salida esperado para cada entrada. Entre las tareas creadas se encuentran copias totales, parciales y tareas completamente originales, además de variar los archivos necesarios

para las entregas (como makefile, por ejemplo) y el uso de comentarios e indentación, las copias totales se disfrazaron cambiando los nombres de variables y el orden de declaración de las funciones.

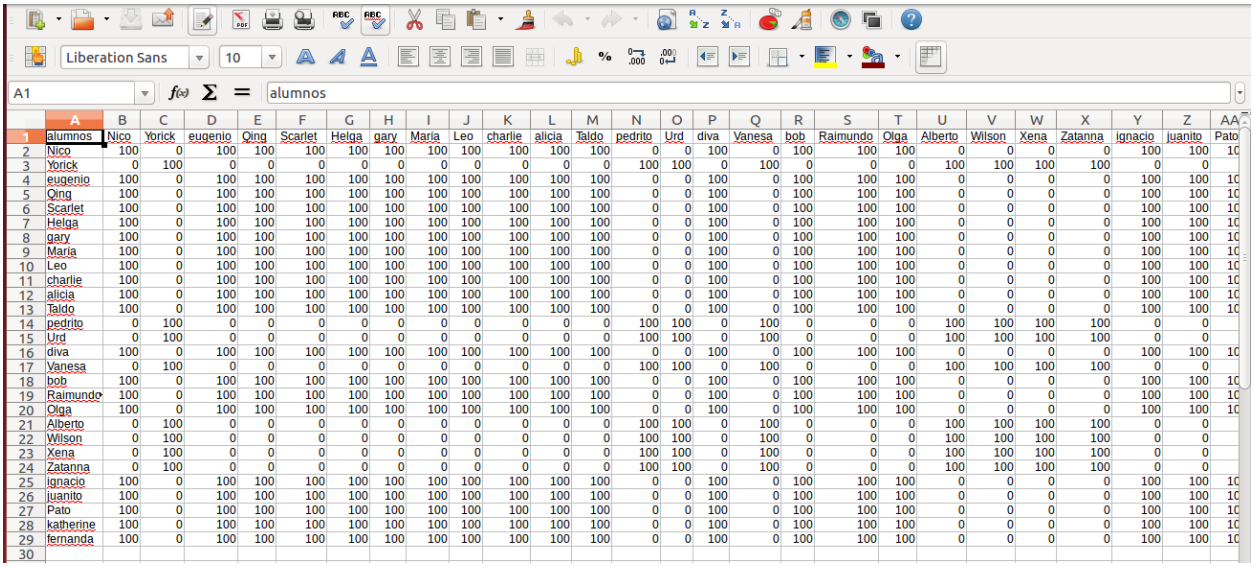
4.1.1 Prueba 1.1: C:

SAC fue ejecutado un total de tres veces sobre la muestra de 28 tareas en lenguaje C, los resultados fueron revisados personalmente y comparados con los esperados, cada vez, se arreglaron los errores encontrados hasta que, en la tercera prueba, no se percibió ningún error en la ejecución. Las figuras 4.1.1.1 y 4.1.1.2 muestran los archivos generados por la tercera y más correcta de las pruebas realizadas, un resumen con tiempos de ejecución y valores mínimos y máximos de similitud en la matriz de plagio puede encontrarse en la tabla 4.1.1. Vale mencionar, SAC encontró satisfactoriamente las tareas copiadas.



	A	B	C	D	E	F	G	H
1	Alumno	p1	p2	p3	p4	p5	Observaciones	
2	Alberto	-1	-1	-1	-1	-1	No hay librerias * No hay makefile * Usa identacion * NO usa comentarios *	
3	Helga	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
4	Leo	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
5	Maria	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
6	Nico	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
7	Olga	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
8	Pato	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
9	Qing	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
10	Raimundo	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
11	Scarlet	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
12	Taldo	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
13	Urd	-1	-1	-1	-1	-1	No hay librerias * No hay makefile * Usa identacion * NO usa comentarios *	
14	Vanesa	-1	-1	-1	-1	-1	No hay librerias * No hay makefile * Usa identacion * NO usa comentarios *	
15	Wilson	-1	-1	-1	-1	-1	No hay librerias * No hay makefile * Usa identacion * NO usa comentarios *	
16	Xena	-1	-1	-1	-1	-1	No hay librerias * No hay makefile * Usa identacion * NO usa comentarios *	
17	Yorick	-1	-1	-1	-1	-1	No hay librerias * No hay makefile * Usa identacion * NO usa comentarios *	
18	Zatanna	-1	-1	-1	-1	-1	No hay librerias * No hay makefile * Usa identacion * NO usa comentarios *	
19	alicia	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
20	bob	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
21	charlie	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
22	diva	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
23	eugenio	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
24	fernanda	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
25	gary	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
26	ignacio	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
27	juanito	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
28	katherine	100	100	100	100	100	No hay librerias * Usa identacion * Usa comentarios *	
29	pedrito	-1	-1	-1	-1	-1	No hay librerias * No hay makefile * Usa identacion * NO usa comentarios *	
30								

Figura 4.1.1.1: Resultados de corrección dinámica, prueba 1.1 (Captura de Pantalla). Fuente: Elaboración propia.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
1	alumnos	Nico	Yorick	eugenio	Qing	Scarlet	Helga	gary	María	Leo	charlie	alicia	Taldo	pedrito	Urd	diva	Vanesa	bob	Raimundo	Olga	Alberto	Wilson	Xena	Zatanna	ignacio	juanito	Pato
2	Nico	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
3	Yorick	0	100	0	0	0	0	0	0	0	0	0	0	100	100	0	100	0	0	0	100	100	100	100	0	0	
4	eugenio	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
5	Qing	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
6	Scarlet	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
7	Helga	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
8	gary	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
9	María	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
10	Leo	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
11	charlie	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
12	alicia	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
13	Taldo	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
14	pedrito	0	100	0	0	0	0	0	0	0	0	0	0	100	100	0	100	0	0	0	100	100	100	100	0	0	
15	Urd	0	100	0	0	0	0	0	0	0	0	0	0	100	100	0	100	0	0	0	100	100	100	100	0	0	
16	diva	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
17	Vanesa	0	100	0	0	0	0	0	0	0	0	0	0	100	100	0	100	0	0	0	100	100	100	100	0	0	
18	bob	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
19	Raimundo	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
20	Olga	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
21	Alberto	0	100	0	0	0	0	0	0	0	0	0	0	100	100	0	100	0	0	0	100	100	100	100	0	0	
22	Wilson	0	100	0	0	0	0	0	0	0	0	0	0	100	100	0	100	0	0	0	100	100	100	100	0	0	
23	Xena	0	100	0	0	0	0	0	0	0	0	0	0	100	100	0	100	0	0	0	100	100	100	100	0	0	
24	Zatanna	0	100	0	0	0	0	0	0	0	0	0	0	100	100	0	100	0	0	0	100	100	100	100	0	0	
25	ignacio	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
26	juanito	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
27	Pato	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
28	katherine	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
29	fernanda	100	0	100	100	100	100	100	100	100	100	100	100	0	0	100	0	100	100	100	0	0	0	0	100	100	10
30																											

Figura 4.1.1.2: Matriz de Plagio generada por la prueba 1.1 (Captura de Pantalla). Fuente: Elaboración Propia.

4.1.2 Prueba 1.2: Python

El mismo proceso anterior fue aplicado a la muestra de 28 tareas en lenguaje Python, requiriendo, al igual que la vez anterior, tres ejecuciones de SAC antes de poder determinar que los datos de salida se conformaban a lo esperado. A pesar de esto, el proceso de corrección dinámica y detección de plagio generaban datos correctos en todas las ejecuciones, si bien en las primeras dos se omitían algunos archivos. Las figuras 4.1.2.1 y 4.1.2.2, al igual que la tabla 4.1.2 presentan los resultados de esta prueba.

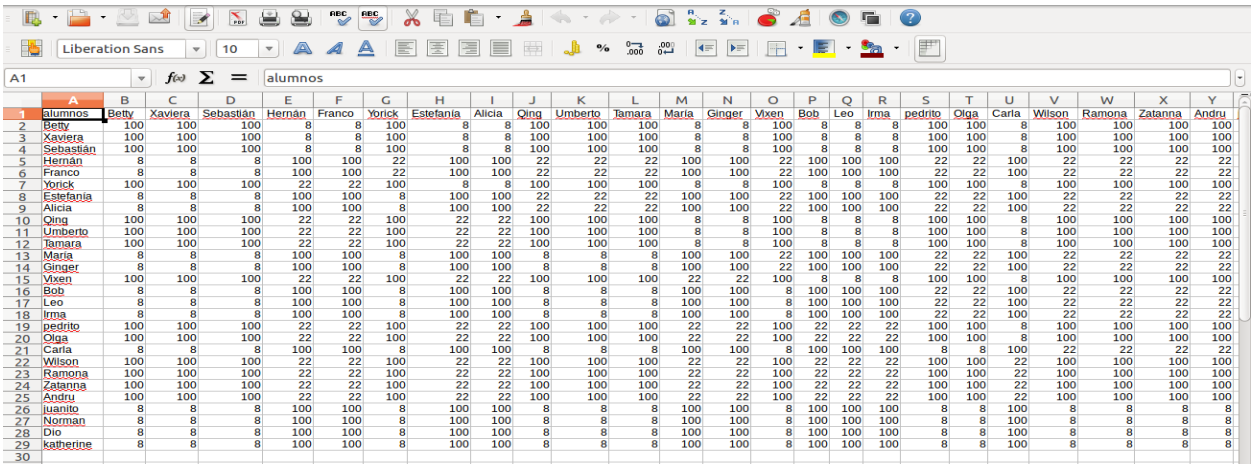
Tabla 4.1.1: Prueba 1.1 en C

Número de Prueba	Tiempo Empleado	Valor Mínimo de Matriz de Plagio	Valor Máximo de Matriz de Plagio	Observaciones
1	2’11’’	0	100	Algunas tareas fueron ignoradas durante la fase de detección de plagio
2	2’ 12’’	0	100	Bug encontrado en estandarización de código
3	2’ 11’’	0	100	Todo según lo esperado.

A1 | Alumno

	A	B	C	D	E	F	G
1	Alumno	p1	p2	p3	p4	p5	Observaciones
2	Alicia	100	0	0	0	0	Usa identacion * NO usa comentarios *
3	Andru	100	100	100	100	100	Usa identacion * NO usa comentarios *
4	Betty	100	100	100	100	100	Usa identacion * NO usa comentarios *
5	Bob	100	0	0	0	0	Usa identacion * NO usa comentarios *
6	Carla	100	0	0	0	0	Usa identacion * NO usa comentarios *
7	Dio	100	0	0	0	0	Usa identacion * NO usa comentarios *
8	Estefanía	100	0	0	0	0	Usa identacion * NO usa comentarios *
9	Franco	100	0	0	0	0	Usa identacion * NO usa comentarios *
10	Ginger	100	0	0	0	0	Usa identacion * NO usa comentarios *
11	Hernán	100	0	0	0	0	Usa identacion * NO usa comentarios *
12	Irma	100	0	0	0	0	Usa identacion * NO usa comentarios *
13	Leo	100	0	0	0	0	Usa identacion * NO usa comentarios *
14	María	100	0	0	0	0	Usa identacion * NO usa comentarios *
15	Norman	100	0	0	0	0	Usa identacion * NO usa comentarios *
16	Olga	100	100	100	100	100	Usa identacion * NO usa comentarios *
17	Qing	100	100	100	100	100	Usa identacion * NO usa comentarios *
18	Ramona	100	100	100	100	100	Usa identacion * NO usa comentarios *
19	Sebastián	100	100	100	100	100	Usa identacion * NO usa comentarios *
20	Tamara	100	100	100	100	100	Usa identacion * NO usa comentarios *
21	Umberto	100	100	100	100	100	Usa identacion * NO usa comentarios *
22	Vixen	100	100	100	100	100	Usa identacion * NO usa comentarios *
23	Wilson	100	100	100	100	100	Usa identacion * NO usa comentarios *
24	Xaviera	100	100	100	100	100	Usa identacion * NO usa comentarios *
25	Yorick	100	100	100	100	100	Usa identacion * NO usa comentarios *
26	Zatanna	100	100	100	100	100	Usa identacion * NO usa comentarios *
27	juanito	100	0	0	0	0	Usa identacion * NO usa comentarios *
28	katherine	100	0	0	0	0	Usa identacion * NO usa comentarios *
29	pedrito	100	100	100	100	100	Usa identacion * NO usa comentarios *
30							

Figura 4.1.2.1: Resultados de Corrección Dinámica prueba 1.2 (Captura de Pantalla). Fuente: Elaboración propia.



alumnos	Betty	Xaviera	Sebastián	Hernán	Franco	Yorick	Estefanía	Alicia	Qing	Umberto	Tamara	María	Ginger	Vixen	Bob	Leo	Irma	pedrito	Olga	Carla	Wilson	Ramona	Zatanna	Andru
Betty	100	100	100	8	8	100	8	8	100	100	100	8	8	100	8	8	8	100	100	8	100	100	100	100
Xaviera	100	100	100	8	8	100	8	8	100	100	100	8	8	100	8	8	8	100	100	8	100	100	100	100
Sebastián	100	100	100	8	8	100	8	8	100	100	100	8	8	100	8	8	8	100	100	8	100	100	100	100
Hernán	8	8	8	100	100	22	100	100	22	22	22	100	100	22	100	100	100	22	22	100	22	22	22	22
Franco	8	8	8	100	100	22	100	100	22	22	22	100	100	22	100	100	100	22	22	100	22	22	22	22
Yorick	100	100	100	22	22	100	8	8	100	100	100	8	8	100	8	8	8	100	100	8	100	100	100	100
Estefanía	8	8	8	100	100	8	100	100	22	22	22	100	100	22	100	100	100	22	22	100	22	22	22	22
Alicia	8	8	8	100	100	8	100	100	22	22	22	100	100	22	100	100	100	22	22	100	22	22	22	22
Qing	100	100	100	22	22	100	22	22	100	100	100	8	8	100	8	8	8	100	100	8	100	100	100	100
Umberto	100	100	100	22	22	100	22	22	100	100	100	8	8	100	8	8	8	100	100	8	100	100	100	100
Tamara	100	100	100	22	22	100	22	22	100	100	100	8	8	100	8	8	8	100	100	8	100	100	100	100
María	8	8	8	100	100	8	100	100	8	8	8	100	100	22	100	100	100	22	22	100	22	22	22	22
Ginger	8	8	8	100	100	8	100	100	8	8	8	100	100	22	100	100	100	22	22	100	22	22	22	22
Vixen	100	100	100	22	22	100	22	22	100	100	100	22	22	100	8	8	8	100	100	8	100	100	100	100
Bob	8	8	8	100	100	8	100	100	8	8	8	100	100	8	100	100	100	22	22	100	22	22	22	22
Leo	8	8	8	100	100	8	100	100	8	8	8	100	100	8	100	100	100	22	22	100	22	22	22	22
Irma	8	8	8	100	100	8	100	100	8	8	8	100	100	8	100	100	100	22	22	100	22	22	22	22
pedrito	100	100	100	22	22	100	22	22	100	100	100	22	22	100	22	22	22	100	100	8	100	100	100	100
Olga	100	100	100	22	22	100	22	22	100	100	100	22	22	100	22	22	22	100	100	8	100	100	100	100
Carla	8	8	8	100	100	8	100	100	8	8	8	100	100	8	100	100	100	8	8	100	22	22	22	22
Wilson	100	100	100	22	22	100	22	22	100	100	100	22	22	100	22	22	22	100	100	22	100	100	100	100
Ramona	100	100	100	22	22	100	22	22	100	100	100	22	22	100	22	22	22	100	100	22	100	100	100	100
Zatanna	100	100	100	22	22	100	22	22	100	100	100	22	22	100	22	22	22	100	100	22	100	100	100	100
Andru	100	100	100	22	22	100	22	22	100	100	100	22	22	100	22	22	22	100	100	22	100	100	100	100
juanito	8	8	8	100	100	8	100	100	8	8	8	100	100	8	100	100	100	8	8	100	8	8	8	8
Norman	8	8	8	100	100	8	100	100	8	8	8	100	100	8	100	100	100	8	8	100	8	8	8	8
Dio	8	8	8	100	100	8	100	100	8	8	8	100	100	8	100	100	100	8	8	100	8	8	8	8
katherine	8	8	8	100	100	8	100	100	8	8	8	100	100	8	100	100	100	8	8	100	8	8	8	8

Figura 4.1.2.2: Matriz de Plagio generada por prueba 1.2 (Captura de Pantalla). Fuente: Elaboración propia.

Tabla 4.1.2: Prueba 1.2 en Python:

Número de Prueba	Tiempo Empleado	Valor Mínimo de Matriz de Plagio	Valor Máximo de Matriz de Plagio	Observaciones
1	7’ 02’’	0	100	Observaciones sobre estilo de código son incorrectas, tiempo de ejecución demasiado largo
2	2’22’’	0	100	Bug encontrado en estandarización de código
3	2’ 23’’	0	100	Todo según lo esperado.

4.2 Prueba con Tareas Reales:

Para esta prueba se utilizaron tareas de la asignatura de Lenguajes de Programación (ILI-253), dictado el segundo semestre de 2016 en la Casa Central de la UTFSM, por el profesor Sven von Brand y ayudantes Eduardo Ramírez y Rodolfo

Castillo. Las tareas facilitadas para la prueba fueron las tareas número 1 y 4, realizadas en lenguajes C y Python respectivamente.

Las tareas usadas para la prueba reciben datos de entrada por teclado de un usuario humano, lo cual las hace inapropiadas para la prueba en modo completo, sin embargo, son perfectas para probar el modo de detección de plagio. Las pruebas se realizaron una vez el semestre estaba concluido, con el fin de que los resultados de la prueba de una aplicación experimental no perjudicaran a los alumnos que cursaban la asignatura.

4.2.1 Prueba 2.1: C:

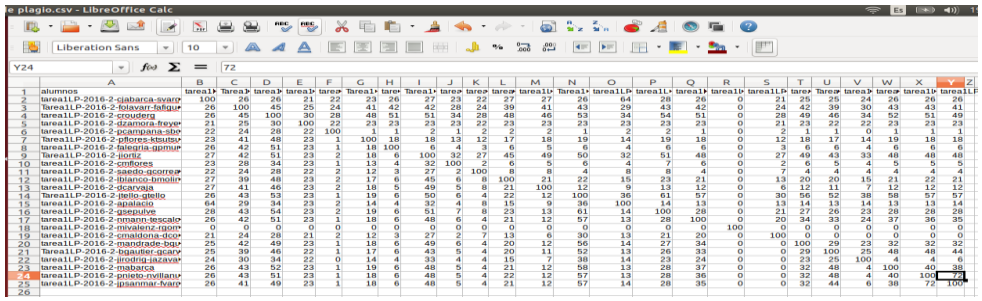
La tarea número 1, programada en lenguaje C y contando con 25 entregas en la muestra recibida, tenía por objetivo crear un juego de combate uno a uno vía texto por consola, en el cual los distintos personajes se cargan dinámicamente desde diferentes archivos de código. Por esta razón, las diferentes entregas presentaban diferentes cantidades de archivos de código, debido a esto, se realiza la predicción de que la posibilidad de copia es relativamente baja. La información entregada por el ayudante Rodolfo Castillo, encargado de la corrección de la tarea confirma que ninguna instancia de plagio fue descubierta para esta tarea. El mismo proceso de las pruebas anteriores se aplicó una vez más, con el fin de depurar errores.

En primera instancia, la matriz de plagio generada por SAC arrojó un valor mínimo de 0% de similitud, y un máximo de 53% de similitud, cantidad que ninguno de los ayudantes consideró como particularmente alta, ciertamente no lo suficiente como para ameritar una interrogación.

Se realizó una segunda prueba, tomando 4 minutos con 4 segundos. El mayor valor de la matriz de plagio subió a un 55%, lo cual, según el ayudante, aun no amerita llamarse plagio.

La tercera prueba realizada se completó en 3 minutos con 56 segundos, la más reciente matriz de plagio muestra un valor máximo de 60% y un mínimo de 0%.

La cuarta y última prueba tomó 5 minutos con 46 segundos, con una similitud mínima de 0% y una máxima de 72%, claramente, tener las líneas correctas para la comparación hace una remarcable diferencia. La tabla 4.2.1 contiene un resumen de las 3 pruebas realizadas, mientras que la figura 4.2.1.1 presenta la matriz de plagio final generada.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	alumnos	tarea1L	Tarea1	tarea1	tarea1	tarea	Tarea1	tare	Tarea1	tarea	tarea	tarea	tarea1L	Tarea1	tarea1L	tarea1L	tarea1	tarea1	tarea1L	tarea	Tarea	tarea1	tarea	tarea1	tarea1LF	
2	tarea1LP-2016-2-cjabarca-svarg	100	26	26	21	22	23	26	27	23	22	27	27	26	64	28	26	0	21	25	25	24	26	26	26	
3	Tarea1LP-2016-2-folavarr-fafgu	26	100	45	25	24	41	42	42	28	24	39	41	43	29	43	42	0	24	42	39	30	43	43	41	
4	tarea1LP-2016-2-crouderg	26	45	100	30	28	48	51	51	34	28	48	46	53	34	54	51	0	28	49	46	34	52	51	49	
5	tarea1LP-2016-2-dzamora-freye	21	25	30	100	22	23	23	23	23	22	23	23	23	23	23	23	0	21	23	22	22	23	23	23	
6	tarea1LP-2016-2-pcampana-sbo	22	24	28	22	100	1	1	2	1	2	2	2	1	2	2	1	0	2	1	1	0	1	1	1	
7	Tarea1LP-2016-2-gflores-ktautsu	23	41	48	23	1	100	18	18	13	12	17	18	19	14	19	18	0	12	18	17	14	19	18	18	
8	tarea1LP-2016-2-falegria-gprmur	26	42	51	23	1	18	100	6	4	3	6	5	6	4	6	6	0	3	6	6	4	6	6	6	
9	Tarea1LP-2016-2-jiortiz	27	42	51	23	2	18	6	100	32	27	45	49	50	32	51	48	0	27	49	43	33	48	48	48	
10	tarea1LP-2016-2-cmflores	23	28	34	23	1	13	4	32	100	2	6	5	6	4	7	6	0	2	6	5	4	5	5	5	
11	tarea1LP-2016-2-saedo-gcorre	22	24	28	22	2	12	3	27	2	100	8	8	4	8	8	4	0	7	4	4	4	4	4	4	
12	tarea1LP-2016-2-jblanco-bmolin	27	39	48	23	2	17	6	45	6	8	100	21	22	15	23	21	0	13	20	20	15	21	22	21	
13	tarea1LP-2016-2-dcarvaja	27	41	46	23	2	18	5	49	5	8	21	100	12	9	13	12	0	6	12	11	7	12	12	12	
14	Tarea1LP-2016-2-jbello-gbello	26	43	53	23	1	19	6	50	6	4	22	12	100	36	61	57	0	30	56	52	38	58	57	57	
15	tarea1LP-2016-2-apalacio	64	29	34	23	2	14	4	32	4	8	15	9	36	100	14	13	0	13	14	13	14	13	13	14	
16	tarea1LP-2016-2-gsepulve	28	43	54	23	2	19	6	51	7	8	23	13	61	14	100	28	0	21	27	26	23	28	28	28	
17	tarea1LP-2016-2-mmann-tescalo	26	42	51	23	1	18	6	48	6	4	21	12	57	13	28	100	0	20	34	33	24	37	36	35	
18	tarea1LP-2016-2-mvalenz-rgon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	
19	tarea1LP-2016-2-cmaldona-dcor	21	24	28	21	2	12	3	27	2	7	13	6	30	13	21	20	0	100	0	0	0	0	0	0	
20	tarea1LP-2016-2-mandrade-bgu	25	42	49	23	1	18	6	49	6	4	20	12	56	14	27	34	0	0	100	29	23	32	32	32	
21	Tarea1LP-2016-2-bgautier-gcarv	25	39	46	22	1	17	6	43	5	4	20	11	52	13	26	33	0	0	29	100	25	48	48	44	
22	tarea1LP-2016-2-jrodrig-jazavar	24	30	34	22	0	14	4	33	4	4	15	7	38	14	23	24	0	0	23	25	100	4	4	6	
23	tarea1LP-2016-2-msbarca	26	43	52	23	1	19	6	48	5	4	21	12	58	13	28	37	0	0	32	48	4	100	40	38	
24	tarea1LP-2016-2-pnieto-nvillanu	26	43	51	23	1	18	6	48	5	4	22	12	57	13	28	36	0	0	32	48	4	40	100	72	
25	tarea1LP-2016-2-jpsanmar-fvarg	26	41	49	23	1	18	6	48	5	4	21	12	57	14	28	35	0	0	32	44	6	38	72	100	
26																										

Figura 4.2.1.1: Matriz de Plagio generada por prueba 2.1 en C. Tarea 1 LP 2016-2 (Captura de Pantalla). Fuente: Elaboración Propia.

Tabla 4.2.1: Pruebas Realizadas en Tarea 1 LP 2016-2:

Número de Prueba	Tiempo Transcurrido	Valor mínimo en matriz de plagio	Valor máximo en matriz de plagio	Errores
1	3’ 23’’	0	53	Observaciones y Comparación de tareas
2	4’ 4’’	0	55	Comparación de tares y reconocimiento de comentarios
3	3’ 56’’	0	60	Estandarización de código.
4	5’ 46’’	0	72	Todo según lo esperado

4.2.2 Prueba 2.2: Python:

Para la segunda prueba real, el profesor von Brand y ayudante Eduardo Ramírez facilitaron la tarea número 4 del semestre. La tarea consistía de realizar una versión simplificada de Google Keep[18]. La muestra contaba con 17 tareas.

El enunciado de la tarea indicaba una fecha de entrega para mediados de noviembre de 2016, lo cual, junto con el menor número de entregas con respecto a la tarea en C, indica una de las últimas tareas del semestre. La relevancia de esto es que en el punto en que los alumnos desarrollaron la tarea, los que aún se mantenían en la asignatura tenían una probabilidad al menos decente de aprobar y, por lo tanto, un manejo adecuado de los conceptos necesarios, sin mencionar una cierta confianza en la capacidad de sus compañeros de trabajo. Todos estos factores contribuyen a la siguiente predicción: Salvo por casos obvios de plagio, la similitud entre las tareas debería ser baja.

Para la primera prueba realizada SAC realizó la detección de plagio en 2 minutos, el valor mínimo encontrado en la matriz de plagio fue de 0%, mientras que el máximo fue de 13%.

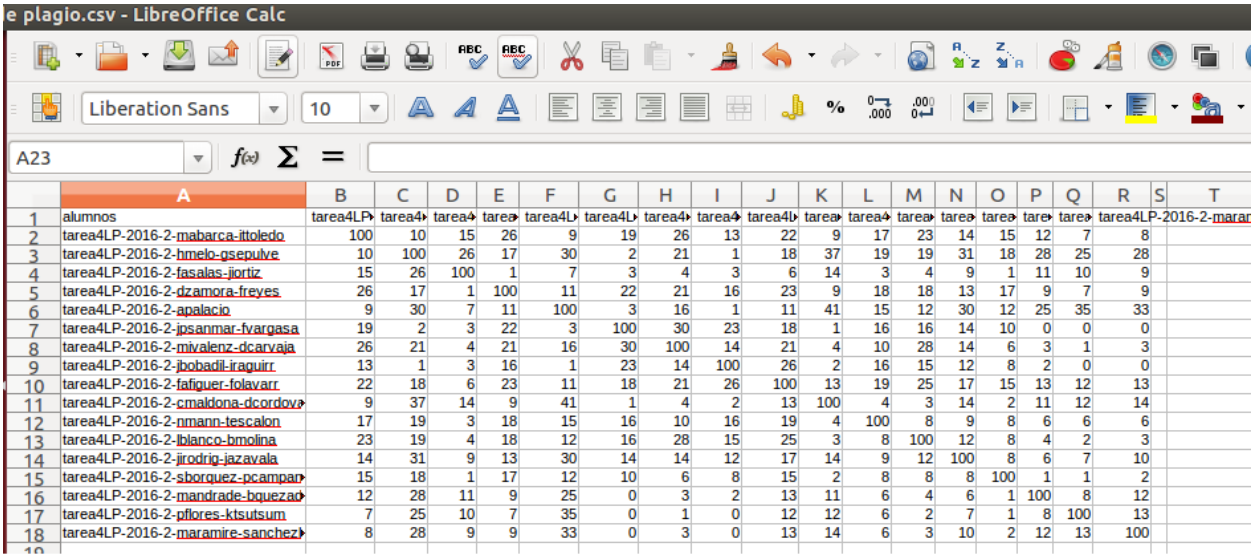
La segunda prueba tomó 2 minutos y 17 segundos, con un rango de similitudes entre el 0% y el 21%.

La tercera prueba con las tareas de Python tomó 2 minutos con 12 segundos, con un mínimo en plagio de 0% y un máximo de 20%.

La cuarta prueba, la cual se culminó en un minuto con 48 segundos, la matriz de plagio mostró un mínimo de 0% de similitud y un máximo de 41%. Se considera que esta prueba fue la más precisa, ya que la mayor cantidad de errores fueron corregidos en el proceso anterior, los resultados de las pruebas aparecen resumidos en la tabla 4.2.2,

[18] https://www.google.com/keep/

mientras que una captura de pantalla de la matriz de plagio de la prueba final puede apreciarse en la figura 4.2.2.1.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	alumnos	tarea4LP	tarea4	tarea4	tarea	tarea4L	tarea4L	tarea4	tarea4	tarea4L	tarea	tarea4	tarea	tarea	tarea	tare	tarea	tarea4LP-2016-2-marar		
2	tarea4LP-2016-2-mabarca-ittoledo	100	10	15	26	9	19	26	13	22	9	17	23	14	15	12	7	8		
3	tarea4LP-2016-2-hmelo-gsepulve	10	100	26	17	30	2	21	1	18	37	19	19	31	18	28	25	28		
4	tarea4LP-2016-2-fasalas-jiortiz	15	26	100	1	7	3	4	3	6	14	3	4	9	1	11	10	9		
5	tarea4LP-2016-2-dzamora-freyes	26	17	1	100	11	22	21	16	23	9	18	18	13	17	9	7	9		
6	tarea4LP-2016-2-apalacio	9	30	7	11	100	3	16	1	11	41	15	12	30	12	25	35	33		
7	tarea4LP-2016-2-jpsanmar-fvargasa	19	2	3	22	3	100	30	23	18	1	16	16	14	10	0	0	0		
8	tarea4LP-2016-2-mivalenz-dcarvaja	26	21	4	21	16	30	100	14	21	4	10	28	14	6	3	1	3		
9	tarea4LP-2016-2-jbobadil-iraguirr	13	1	3	16	1	23	14	100	26	2	16	15	12	8	2	0	0		
10	tarea4LP-2016-2-fafiguer-folavarr	22	18	6	23	11	18	21	26	100	13	19	25	17	15	13	12	13		
11	tarea4LP-2016-2-cmaldona-dcordova	9	37	14	9	41	1	4	2	13	100	4	3	14	2	11	12	14		
12	tarea4LP-2016-2-nmann-tescalon	17	19	3	18	15	16	10	16	19	4	100	8	9	8	6	6	6		
13	tarea4LP-2016-2-lblanco-bmolina	23	19	4	18	12	16	28	15	25	3	8	100	12	8	4	2	3		
14	tarea4LP-2016-2-jirodrig-jazavala	14	31	9	13	30	14	14	12	17	14	9	12	100	8	6	7	10		
15	tarea4LP-2016-2-sborquez-pcampar	15	18	1	17	12	10	6	8	15	2	8	8	8	100	1	1	2		
16	tarea4LP-2016-2-mandrade-bquezad	12	28	11	9	25	0	3	2	13	11	6	4	6	1	100	8	12		
17	tarea4LP-2016-2-pflores-ktsutsum	7	25	10	7	35	0	1	0	12	12	6	2	7	1	8	100	13		
18	tarea4LP-2016-2-maramire-sanchez	8	28	9	9	33	0	3	0	13	14	6	3	10	2	12	13	100		

Figura 4.2.2.1: Matriz de Plagio generada por prueba 2.2 en Python. Tarea 4 LP 2016-2 (Captura de Pantalla). Fuente: Elaboración Propia.

Tabla 4.2.2: Pruebas realizadas en tarea 4 de LP, 2016-2.

Número de prueba	Tiempo Transcurrido	Valor mínimo de la matriz	Valor Máximo de la matriz	Observaciones
1	2’ 00’’	0	13	Problema con detección de comentarios
2	2’ 17’’	0	21	Problema con procesamiento de comentarios
3	2’ 32’’	0	20	Problema con estandarización de nombres
4	1’ 48’’	0	41	Todo según lo esperado.

4.3 Análisis de Resultados:

En la literatura establecida, muchas métricas se mencionan a la hora de evaluar el desempeño de un sistema de corrección automatizado, muchas de ellas dicen relación con el aspecto pedagógico de la labor realizada, sin embargo, fue considerado un riesgo innecesario el someter el trabajo de los estudiantes a corrección bajo un sistema aún experimental, en especial dada la cantidad de errores que se encontraron una vez el sistema creado entró en contacto con tareas reales. Dado lo anterior, una prueba en un ambiente real de corrección, durante el transcurso de un semestre habitual, no se ha realizado al momento de escribir el presente documento. En segundo lugar, la evaluación pedagógica del sistema escapa al espectro de una memoria de ingeniería, ya que requeriría experticia que escapa a las habilidades impartidas por la universidad en la carrera de Ingeniería Civil en Informática.

Un análisis técnico del desempeño de SAC es mucho más factible que un análisis pedagógico, al igual que mucho más rápido de efectuar, ya que no requiere del transcurso de uno o, idealmente, varios semestres. Para un análisis desde este punto de vista, las dos métricas más obvias serían tiempo y efectividad.

Desde el punto de vista del tiempo, SAC realiza un trabajo bastante adecuado, al enfrentarse a programas pequeños, es capaz de realizar varias pruebas en una muestra de tamaño curso en menos de cinco minutos, mientras que puede realizar detección de plagio en programas grandes, de nuevo en una prueba de tamaño curso en menos de 10 minutos, entregando información detallada sobre la calidad de la entrega, la similitud entre las tareas y la efectividad del programa. Un lector observador habrá notado que, a pesar de usar una muestra más grande y un proceso más largo, las pruebas sobre tareas ficticias (creadas por el memorista) se terminaron en tiempos mucho más cortos que las pruebas en tareas reales, en las cuales sólo se realizó la detección de plagio, esto se debe al modo de operación de los procesos de transcripción, estandarización y comparación de tareas, como estos procesos se realizan línea por línea, el número de líneas de cada programa se vuelve el factor decisivo en determinar el tiempo necesario para la detección de plagio, mientras que el número de pruebas es el factor determinante

para el proceso de corrección automática. Finalmente, sobre el tiempo de ejecución, vale notar que no ha sido posible ejecutar una comparación entre SAC y otros programas disponibles para la corrección automática de tareas, sin embargo, el trabajo de corrección, que al ayudante le tomaría una semana o más realizar sin asistencia, se realiza en menos de media hora, liberando una considerable cantidad de tiempo.

Desde el punto de vista de la efectividad, SAC es capaz de generar observaciones certeras sobre el formato de entrega y el estilo de programación de un alumno, entregando información sobre el uso de bibliotecas, makefile, comentarios, indentación y la generación correcta de archivos tales como ejecutables y archivos de salida, es capaz de evaluar la similitud porcentual de la salida generada por el alumno con respecto de la esperada para varios casos de prueba diferentes y compilar toda esta información en un archivo de fácil acceso para el corrector de la tarea. En cuanto a la detección de plagio, SAC es capaz de detectar copia entre dos programas a pesar de cambios de nombres en variables y funciones, cambio de orden en la declaración de funciones y la presencia o ausencia de comentarios que pudieran entorpecer la lectura de la tarea. Al comparar función con función, línea por línea, SAC puede incluso detectar copia en caso de que dos tareas tengan una diferente cantidad de líneas de código. Por lo cual, se concluye que el objetivo de detección de plagio se ha cumplido.

Ahora bien, SAC es capaz de detectar la ausencia de múltiples prácticas que se espera del alumno que domine, sin embargo, cuando llega el momento de evaluar a los alumnos que cumplen y la medida en que lo hacen, un corrector debería confiar en su propia experticia y no en el sistema, ya que varios aspectos de la calidad del código son más detectables por ojos humanos que por una máquina. Un ejemplo de lo anterior es el proceso de nombrar variables y funciones, sin entregarle al sistema contexto sobre el dominio del problema, este aspecto es imposible de evaluar para un sistema automático.

Sobre el tema de la detección de plagio queda una pregunta por responder: ¿En qué punto empieza el plagio y termina la reutilización de código? Esta pregunta, vital como pueda ser, se deja a la experticia de los profesores y ayudantes, ya que es un análisis de carácter pedagógico y escapa al espectro del proyecto. La cuestión de

reutilización de código versus plagio no es una trivial, hablando en términos de porcentajes de similitud, dibujar la línea en porcentajes muy bajos conlleva el riesgo de obtener una gran cantidad de falsos positivos, sin mencionar que es poco realista, ya que si existe un número finito de algoritmos que resuelven un problema, es natural pensar que dos o más tareas tengan algún grado de similitud sin que haya ningún tipo de copia por parte de los alumnos responsables. El otro extremo del problema es igualmente, si es que no más, riesgoso, ya que un porcentaje de similitud demasiado alto dejaría pasar desapercibidos los intentos de copia, lo cual es injusto hacia los alumnos que sí trabajaron duro por entregar sus tareas y además es perjudicial para el alumno realizando el fraude académico, ya que se le permite avanzar en la carrera sin dominar realmente los contenidos y conceptos que las tareas pretenden evaluar. Es por esta razón que SAC no entrega una lista de alumnos sospechosos de copia, sino una matriz de similitudes entre las tareas entregadas, queda a discreción del profesor o ayudantes el discernir si una similitud de, por ejemplo, un 80% cae en fraude académico o en simple reutilización de código.

Capítulo 5: Conclusiones

En el presente capítulo se presentan los aprendizajes logrados en el desarrollo del proyecto, se evalúa el cumplimiento de los objetivos planteados al principio del mismo y se entregan ejemplos del posible futuro trabajo basado en este proyecto.

5.1 Evaluación del Cumplimiento de los Objetivos:

La evaluación de se realizará comenzando por cada objetivo específico, para así presentar una mejor visión del cumplimiento del objetivo general.

- Objetivo 1: "Evaluar la Factibilidad de una conexión entre el sistema de corrección y la plataforma Moodle del Departamento de Informática. Implementando dicha conexión de ser posible". Como se ha expuesto en los capítulos dos y tres del presente documento, una conexión con Moodle no es solo posible o factible, sino que se ha realizado para sistemas parecidos, sin embargo, tomando en consideración que el dominio del uso de repositorios de código es importante en el ámbito profesional y que una extensión de Moodle animaría a profesores y ayudantes a abandonar la idea de enseñar el uso de repositorios desde temprano en la carrera, se decidió que no es recomendable convertir SAC en una extensión de Moodle, sin embargo, no es por ningún motivo imposible.
- Objetivo 2: "Desarrollar una aplicación capaz de compilar y ejecutar programas para algún lenguaje de programación por definir, evaluando el desempeño del programa según los datos de salida esperados". Este objetivo en particular no solo se ha cumplido, sino que se ha excedido es estándar impuesto al principio. No solo es SAC capaz de compilar y ejecutar tareas dada una entrada y salida esperada, sino que es capaz de hacer esto para dos lenguajes diferentes, siendo estos el lenguaje introductorio que la UTFSM utiliza en su ramo de Programación, Python, y el lenguaje más usado en asignaturas del Departamento de Informática, C. Adicionalmente, un sistema de detección de plagio fue implementado, el cual

hace de SAC no sólo útil para tareas con una entrada y salida específicas entregadas por archivos, sino para tareas con entrada y salida entregadas por usuarios en tiempo real.

- Objetivo 3: “Evaluar tareas de programación según su apego a las buenas prácticas definidas según el lenguaje de programación elegido”. En este objetivo, las limitaciones del computador con respecto del hombre se hacen presentes, si bien, SAC puede detectar el uso de bibliotecas, comentarios, indentación, makefile y varias otras prácticas, el sistema diseñado no tiene forma de saber si los comentarios son pertinentes, o si los nombres de variables y funciones son significativos respecto del dominio del problema, sin embargo, SAC puede aligerar la carga de trabajo del corrector señalando cuáles alumnos no cumplen con las buenas prácticas, dejando en manos humanas el determinar cuáles alumnos cumplen con las prácticas de una forma útil y correcta.
- Objetivo 4: “Implementar medidas de seguridad que protejan al corrector de código malicioso o incorrecto”. Una de las más importantes medidas de seguridad adoptadas por SAC es no ser una extensión de Moodle, limitando el efecto destructivo de un posible ataque a un computador específico y no a los servidores del Departamento de Informática. En una escala más local, dado que SAC tiene la capacidad de comparar y determinar la similitud entre dos programas, comparar los programas entregados contra programas dañinos arrojaría algunas luces sobre qué programas no ejecutar, incluso fallando esta capa de protección, se podría correr SAC en modo plagio y como un usuario sin privilegios, lo cual protegería el sistema utilizado de código malicioso.
- Objetivo General: “Desarrollar una arquitectura segura y escalable que permita la corrección automática de tareas de programación para el Departamento de Informática de la Universidad Técnica Federico Santa María”. Con la información entregada en la lista anterior, se puede decir que el sistema creado cumple con las características de seguridad y escalabilidad, cumpliendo con el objetivo general del proyecto.

5.2 Trabajo Futuro:

Una de las más obvias direcciones en las que podría mejorarse SAC es la inclusión de más lenguajes de programación, idealmente, todos los lenguajes enseñados en la asignatura Lenguajes de Programación, siendo Java uno de los más importantes. El remover limitaciones en los tipos de tareas que SAC pueda evaluar es también deseable, sin embargo, por lo menos un equipo de investigación afirma que la entrada/salida establecida es absolutamente necesaria para el funcionamiento de un sistema de corrección automática (2).

Las diferentes aplicaciones del SAC también resultan interesantes de contemplar, desde un sistema que automáticamente corrige y entrega resultados vía Moodle como lo es CTPracticals (14) hasta plataformas de ejercicios en tiempo real tales como CodeRunner[19] o HackerRank[20]. Otro posible avance de SAC es la implementación de la llamada corrección estática, es decir, enseñarle al sistema a leer el código y evaluar la comprensión y aplicación de los conceptos que se buscaba enseñar con la tarea.

La posibilidad de entregar retroalimentación a los estudiantes con respecto a su desempeño, más allá de su calificación, es una mejora ideal para un sistema de corrección automática, ya que no solo remueve algo de la carga laboral del educador, sino que además ayuda a mejor orientar el aprendizaje del alumno.

La operación de SAC podría resultar bastante más lenta dependiendo de la cantidad de tareas a corregir, por lo que un ejercicio de paralelización de la operación del sistema sería interesante de abordar. Probar SAC en un ambiente real, como podrían ser los laboratorios de programación, es una buena oportunidad para intentar esto, sin mencionar una buena prueba de fuego con tareas reales en tiempo real.

Finalmente, la evaluación de SAC comparado con otros sistemas es un tema digno de investigar, comparaciones entre SAC y sistemas de corrección como Web-Cat

[19] http://coderunner.org.nz/

[20] https://www.hackerrank.com/school#video

resultan relevantes para determinar la efectividad y completitud de SAC como corrector automático, así como la comparación entre la corrección automática contra la manual. Comparaciones con otros sistemas de detección de plagio, como puede ser Sherlock (27), también resultan interesantes. Otro posible tema de investigación es una evaluación de seguridad de las aplicaciones en línea detalladas en el capítulo 2, comparadas con una aplicación fuera de línea como SAC.

5.3 Aprendizajes Logrados:

Una conclusión que tal vez suene decepcionante es que, después de varios meses de trabajar en un proyecto de titulación de ingeniería civil en informática, se puede decir que los ingenieros no son profesores, por lo que todo el trabajo realizado en este proyecto se abordó de un punto de vista técnico, más que pedagógico, la formación entregada por la UTFSM no entrega (ni debe necesariamente entregar) las herramientas para una evaluación pedagógica del trabajo realizado.

Habiendo dicho esto, un sistema de ayuda a la enseñanza fue desarrollado, con el fin de automatizar en un cierto grado la labor de la corrección de tareas en el Departamento de Informática y construir los cimientos de un trabajo futuro en el área de la corrección automática, no sólo en la UTFSM, sino también a nivel nacional, ya que durante la investigación realizada para el presente proyecto, ninguna publicación por investigadores nacionales fue encontrada en la literatura disponible. Las pruebas realizadas sobre el SAC se abordaron desde el punto de vista de requerimientos cumplidos, ya que el tiempo necesario para probar la existencia de una mejora en el aprendizaje de los alumnos con la inclusión del SAC correspondería a varios semestres, sin mencionar las razones ya expuestas. Con respecto a estos aspectos técnicos, las pruebas han resultado satisfactorias, SAC espera a ser implementado como ayuda a profesores y ayudantes y ser sometido a pruebas por parte de investigadores más capacitados con respecto a sus beneficios pedagógicos.

Una conclusión interesante que se puede sacar del trabajo realizado es la falta de investigación en el campo de la corrección automática proveniente de profesionales chilenos, una escasez que, se espera, este proyecto pueda comenzar a remediar.

Referencias

1. **Universidad Técnica Federico Santa María.** Página Web. Visitada el 17 de Mayo de 2017.

2. *Automated Assessment of Programming Assignments.* **Pieterse, Vreda.** 2013. Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research. págs. 45-56 .

3. *Improving student performance by evaluating how well students test their own programs.* **Eduards, S.H.** 3, Septiembre de 2003, ACM Journal of Educational Resources in Computing,, Vol. 3.

4. *An Experience on Ada Programming Using On-Line Judging.* **Montoya-Dato F.J., Fernández-Alemán J.L., García-Mateos G.** (2009) In: Kordon F., Kermarrec Y. (eds) Reliable Software Technologies – Ada-Europe 2009. Ada-Europe 2009. Lecture Notes in Computer Science, vol 5570. Springer, Berlin, Heidelberg

5. *Automatic marking with Sakai.* **Suleman, Hussein.** Wilderness : s.n., 2008. SAICSIT '08 Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology. págs. 229-236 .

6. *An automated feedback system for computer organization.* **Chen, PM.** 2, s.l. : IEEE Press, Mayo de 2004, IEEE Trans. on Educ., Vol. 47, págs. 232--240.

7. *On automated grading of programming assignments in an academic institution.* **Cheang, Brenda, y otros.** s.l. : Elsevier ltd., 2003, Computers & Education, Vol. 41, págs. 121-131.

8. *The BOSS online submission and assessment system.* **Joy, M., Griffiths, N. y R.Boyatt.** 3, s.l. : ACM, 2005, Journal on Educational Resources in Computing (JERIC), Vol. 5.

9. **Ihantola, Petri, y otros.** Review of Recent Systems for Automatic Assessment of Programming Assignments. *Proceedings of the 10th Koli Calling International Conference on Computing Education Research.* s.l. : ACM, 2010, págs. 86--93.

10. **Caiza, Julio y del Alamo, José.** *PROGRAMMING ASSIGNMENTS AUTOMATIC GRADING: REVIEW OF TOOLS AND IMPLEMENTATIONS.* 2013.

11. *Web-cat: automatically grading programming assignments.* **Eduards, S.H. y Quiñones, M.A.** Madrid, España : ACM, 2008. Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE '08.

12. *”mailing it in”: email-centric automated assessment.* **Sant, J.A.** Paris, Francia : ACM , 2009. ITiCSE '09 Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education. págs. 308-312 .

13. *Educomponents: Experiences in e-assessment in computer science education.* **Amelung, M., Forbrig, P. y Rösner, D.** New York, NY. : ACM Press, 2006. ITiCSE ’06: Proceedings of the 11th annual conference on Innovation and technology in computer science. págs. 88-92.

14. *A new Moodle module supporting automatic verification of VHDL-based assignments.* **Gutiérrez, Eladio, y otros.** 2, OXford, UK. : Elsevier Science Ltd, 2010, Computers & Education, Vol. 54, págs. 562-577 .

15. *Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses.* **Spacco, Jaime, y otros.** New York, NY. : ACM Press., 2006. ITiCSE ’06: Proceedings of the 11th annual conference on Innovation and technology in computer science. págs. 13-17.

16. *Towards generic and Flexible Web Services for E-Assessment.* **Amelung, Mario, Forbrig, Peter y Rösner, Dietmar.** Madrid, España. : ACM, 2008. ITiCSE '08 Proceedings of the 13th annual conference on Innovation and technology in computer science education. págs. 219-224 .

17. **Patil, Ashlesha.** *Automatic Grading of Programming Assignments.* 2010. Tesis de Magister.

18. **Horstmann, Cay.** *LabRat.*

19. *Ability-training-oriented automated assessment in introductory programming course.* **Wang, Tiantian, y otros.** 1, Oxford, UK. : Elsevier Science Ltd, Enero de 2011, Computers & Education, Vol. 56, págs. 220-226.

20. *AutoLEP: An Automated Learning and Examination System for Programming and its Application in Programming Course.* **Wang, Tiantian, y otros.** Wuhan, Hubei, China : IEEE, 2009. 2009 First International Workshop on Education Technology and Computer Science.

21. *Semantic similarity-based grading of student programs.* **Wang, Tiantian, y otros.** 2, 2007, Information and Software Technology, Vol. 49, págs. 99-107.

22. **Abd Rahman, Khirulnizam, Ahmad, Syarbaini y Nordin, Md Jan.** *The Design of an Automated C Programming Assessment Using Pseudo-code Comparison Technique.* 2007.

23. *Using a linux security module for contest security.* **Merry, B.** 3, 2009, Olympiads in Informatics, págs. 67–73.

24. **Chen, M.Y., y otros.** *Design and applications of an algorithm benchmark system in a computational problem solving environment.* 2006.

25. *The toolbox for local and global plagiarism detection.* **Butakov, Sergey y Scherbinin, Vladislav.** 4, Oxford, UK. : Elsevier Science Ltd, 2008, Computers & Education, Vol. 52, págs. 781-788.

26. *Winnowing: Local Algorithms for Document Fingerprinting.* **Schleimer, Saul, Wilkerson, Daniel S. y Aiken, Alex.** San Diego, CA : s.n., 2003. Proceedings of SIGMOD conference 2003. págs. 76-85.

27. *Plagiarism in Programming Assignments.* **Joy, Mike y Luck, Michael.** 2, Piscataway, NJ, USA : IEEE Press, mayo de 1999, IEEE Transactions on Education, Vol. 42, págs. 129--133.

28. *Identification of Program Similarity in Large Population.* **Whale, G.** 2, Oxford, UK : Oxford University Press, abril de 1990, Comput. J., Vol. 33, págs. 140--146.

29. *Detection of Similarities in Student Programs: YAP'ing maybe preferable to Plague'ing.* **Wise, Michael J.** Kansas City, Missouri, USA : ACM, 1992. IGCSE '92 Proceedings of the twenty-third SIGCSE technical symposium on Computer science education. Vol. 24, págs. 268--271.

30. **Moodle Pty Ltd.** *Moodle.* 2001.

31. *IMPROVED ASSIGNMENTS MANAGEMENT IN MOODLE ENVIRONMENT.* **Jelemenská, Katarína y Čičák, Pavel.** Valencia, España. : s.n., 2012. INTED2012 Proceedings: International Technology, Education and Development Conference. págs. 1809-1816.