PLATAFORMA DE EVALUACIÓN AUTOMÁTICA DE PROGRAMAS

Jorge López Reguera[1] Cecilia Hernández Rivas[1] Yussef Farran Leiva[1]
jorlopez@udec.cl cecihernandez@udec.cl yfarran@udec.cl

[1]Depto de Ingeniería Informática y Ciencias de Computación
Facultad de Ingeniería - Universidad de Concepción

Resumen

Este artículo describe una plataforma de evaluación automática de programas y presenta los resultados obtenidos en su utilización para las sesiones prácticas (con alumnos frente al computador) de las asignaturas de Lenguaje de Programación y Redes de Computadores. Se describe la arquitectura de la plataforma así como también la estrategia utilizada en su utilización en las asignaturas mencionadas. Se presenta un análisis de los resultados y se discuten los aspectos en los cuales la plataforma apoya la docencia. Finalmente, se exponen las limitaciones de esta forma de evaluación y se esboza una estrategia para mejorar la metodología de enseñanza de la programación.

Palabras claves: Prueba de Programas, Evaluación Automática.

Abstract

This paper describes an automatic program evaluation platform (framework) and presents the results we obtained after using it in practical sessions (with students in front of the computer) in the subjects of Programming Languages and Computer Networks. We describe the platform architecture and the strategy used in the subjects mentioned above. We present an analysis of the results and a discussion of the aspects that the platform provides to improve the teaching and learning experience. Finally, we mention the limitations of this evaluation technique and we propose a strategy to improve the methodology in teaching how to program.

1. Introducción

El problema de trabajar con cursos de programación numerosos es muy común en la actualidad [2]. Estos cursos imponen una demanda adicional en comparación con los cursos tradicionales que están diseñados para soportar principalmente evaluaciones escritas.
Enseñar a construir programas es una tarea que incluye, entre otros aspectos, la comprobación que el alumno es capaz de producir un programa que cumple con parámetros específicos. La metodología de evaluación en cursos de programación, que hemos usado hasta ahora, incluye:

a) *Pruebas escritas*. Este tipo de evaluaciones consisten en una inspección visual de las respuestas centrando el esfuerzo del evaluador en el análisis de la estructura del programa. Sin embargo, al aumentar el tamaño de los cursos, se tiende a limitar el tamaño de los programas y a disminuir la frecuencia de aplicación. Una prueba de este tipo típicamente toma entre una y dos horas, más el tiempo empleado en la revisión, la cual es proporcional al número de alumnos.
b) *Tareas de programación*. Este tipo de evaluaciones permite enfrentar al alumno con la construcción de programas más complejos. Sin embargo, al aumentar el número de alumnos se debe buscar un balance entre el tamaño y el número de las tareas, llegando en muchos casos a asignarlas a grupos de alumnos, lo cual no nos permite garantizar que todos trabajaron equitativamente. El tiempo total para una evaluación de este tipo incluye tanto el tiempo que le toma a los alumnos realizarla (proporcional al tamaño) más el tiempo de revisión (proporcional al tamaño y al tipo de evaluación). Las evaluaciones suelen incluir la ejecución de los programas y la inspección visual de su estructura.
c) *Programación frente al computador*. Es la evaluación que mejor demuestra si un alumno está capacitado para crear programas, sin embargo es necesario contar con una plataforma computacional adecuada al número de alumnos de cada curso o en su defecto, dividir el curso para realizar la prueba. En este caso la mayor dificultad con la que nos hemos encontrado radica en el hecho que los alumnos trabajan en paralelo y cada evaluador

(frecuentemente una sola persona) debe ocuparse de varios alumnos a la vez, es decir, el evaluador se ve forzado a realizar la revisión de las respuestas en forma concurrente y en un período de tiempo relativamente corto.

Las principales deficiencias de la metodología descrita arriba incluyen:

1. Imposibilidad de realizar controles frecuentes, en base a objetivos específicos, en cursos numerosos.
2. Posibilidad de plagio de tareas de programación entre grupos
3. Dificultades en la evaluación individual o personalizada
4. Dificultades en identificar/apoyar a los alumnos con mayores problemas de aprendizaje
5. Dificultad de evaluar imparcialmente
6. Proceso de revisión involucra mucho tiempo en cursos numerosos.

Consideramos que un sistema de evaluación idealmente debiera contar con las siguientes características:

1. Someter al alumno a una situación similar a lo que experimentará en su actividad profesional [8].
2. Medir todos los aspectos deseados y con el mínimo error posible.
3. Medir con una granularidad más fina (frecuencia de evaluaciones) de acuerdo a objetivos específicos.
4. Entregar los resultados automáticamente para que los alumnos y profesores puedan ajustar su esfuerzo y reforzar los aspectos deficitarios a tiempo.
5. Garantizar que cada alumno sea capaz de construir programas de acuerdo a lo enseñado en clases, aplicando evaluación individual.
6. Adecuar el esfuerzo que involucra realizar la evaluación a cursos numerosos por parte del profesor, de forma que se concentre más en la calidad que en la cantidad.
7. Entregar una calificación imparcial, a cada alumno, al finalizar el curso.

Las deficiencias de la metodología actual y las características que consideramos debería tener un sistema de evaluación donde el desarrollo práctico es importante nos motivó para la construcción de una plataforma que apoye el proceso de evaluación (tipo c), con el propósito de lograr los siguientes objetivos:

a) Comprobar que el alumno puede crear un programa en un determinado período limitado de tiempo.
b) Obtener los resultados de evaluación oportunamente.
c) Evaluar con la frecuencia requerida por los objetivos parciales de cada curso.
d) Eliminar la posibilidad de que se pueda aprobar el curso eludiendo realizar el trabajo práctico y por lo tanto no desarrollando la competencia necesaria.
e) Permitir realizar las evaluaciones de un conjunto de alumnos en paralelo.
f) Obtener información más profunda de la situación general del curso y particular de cada alumno.

Para lograr los objetivos anteriores, fue necesario que la plataforma cumpliera con el requisito de revisar automáticamente los programas de los alumnos, en otras palabras realizar “prueba de software”.

Este artículo está organizado de manera que en la sección 2 revisamos los fundamentos de la prueba de programas, en la sección 3 mostramos plataformas con algún grado de automatización en la prueba de programas, en la sección 4 describimos la arquitectura de nuestra plataforma, en la sección 5 describimos los criterios de diseño del módulo probador y dos casos de aplicación de la prueba de programas, en la sección 6 realizamos un análisis de los resultados de la aplicación de la plataforma y una encuesta realizada a los estudiantes sometidos a estas pruebas, para finalizar en la sección 7 con las conclusiones y la sección 8 de trabajos futuros.

2. Fundamentos

Según [5], la “prueba de software” consiste en confirmar la calidad del software con métodos que se puedan aplicar de forma económica y efectiva, tanto a grandes como a pequeños programas”. [1] clasifica la prueba de software en “verificación” y “validación”. La verificación o “prueba de programas” consiste en un conjunto de actividades que aseguran que el software implementa correctamente una función específica. La validación involucra un conjunto de actividades que aseguran que el software construido satisface los requerimientos del cliente.

Las pruebas de programas se pueden realizar tanto en forma estática como en forma dinámica (ejecución del programa). Las pruebas estáticas consisten en inspección del código fuente de un programa o mediante un sistema de software. Las pruebas dinámicas pueden ser de tipo "caja negra", que se aplican sobre la interfaz del software, o de tipo "caja blanca", que se refieren al examen de los detalles procedurales del programa.

En [7], se definen los siguientes criterios para la Adecuación de la Prueba de Programas:
a) *Cobertura de sentencias*. Requiere generar casos de prueba para ejecutar todas las sentencias de un programa.
b) *Cobertura de transferencias de control*. Similarmente, este criterio requiere que se prueben todas las transferencias de control del programa.
c) *Cobertura de Trayectorias*. Este criterio requiere que todas las trayectorias de ejecución desde la entrada del programa hasta su finalización, sean ejecutadas.
d) *Adecuación de mutaciones*. Consiste en agregar artificialmente errores al programa y comprobar si son detectados por la prueba.

Los objetivos de la prueba de programa, según se desprende de lo afirmado por Dijkstra en 1972 y de las reglas establecidas por [6], sólo pueden demostrar la existencia de un conjunto de defectos específicos en el software, pero no la ausencia total de defectos. Actualmente la prueba de programas involucra a una gran variedad de objetivos, estrategias y niveles de detalles, sin embargo todavía se puede considerar ligada a la habilidad para detectar defectos como de concluye en [7].

En [3] se propone un conjunto de principios esenciales para probar programa siguiente:
a) *Especificación*. Define lo que significa un programa correcto.
b) *Premeditación*. Debe ser un proceso sistemático.
c) *Repetibilidad*. El proceso de prueba y sus resultados deben ser repetibles e independientes del probador.
d) *Auditabilidad*. El proceso de prueba debe dejar una huella auditable.
e) *Economía*. El proceso de prueba no debe requerir excesivos recursos.

La estrategia para probar programas de [4] propone dividir el esfuerzo de pruebas dinámicas al menos en dos ejecuciones, la prueba nominal o de condiciones ideales, y la prueba de manejo de errores.

3. Trabajos relacionados

Un caso de plataforma de revisión automática es la que fue creada en ACM Programming Contest (competencia de programación). Sin embargo, el objetivo de la plataforma ACM es determinar quien resuelve más problemas en la menor cantidad de tiempo, hace un análisis de "caja negra" de los programas, para ello les entrega datos específicos de prueba como entrada y luego determina si la ejecución produce las salidas esperadas.
Otro sistema que incluye revisión automática de programas es descrito en [4], el cual está orientado a revisar tareas de programación con la técnica de prueba del tipo caja negra y proporciona retroalimentación inmediata al estudiante. El sistema TRY, [Reek 1989], sólo proporciona una infraestructura para la revisión de programas con retroalimentación inmediata al estudiante, pero sólo de los resultados de la compilación y no proporciona evaluación automática de la ejecución.
A diferencia de los sistemas anteriores, nuestra plataforma se orienta a la realización de pruebas de corta duración (no superior a dos horas.) frente al computador, que se aplican simultáneamente a un conjunto de alumnos. Sin embargo, nuestra plataforma también puede ser utilizada para la evaluación automática de tareas de programación, aunque aún no la hemos utilizado en ese contexto. Las características de la plataforma son las siguientes:

a) Realiza calificaciones en línea y va entregando resultados parciales durante la prueba.
b) Incluye alarmas que detectan posibles casos de personas respondiendo por otras.
c) Cuenta con una interfaz que permite monitorear el estado de avance de los alumnos.
d) A cada programa de la prueba se le puede exigir que logre una o varias metas, con el respectivo puntaje asociado.
e) Cada programa puede ser revisado varias veces sin alterar el puntaje obtenido al haberse ya alcanzado metas en revisiones previas.
f) Cuenta con control de tiempo de ejecución para detectar ciclos sin fin, incluye el manejo de los errores de acceso a memoria dentro de un programa permitiendo probar todas las metas sin que se interrumpa la ejecución.

g) Registra todos los programas que los alumnos envían para ser probados asociados a una estampa de tiempo.

4. Arquitectura de la plataforma

En el diseño modular de la plataforma se define una interfaz para permitir que el profesor de la asignatura sea el responsable de la construcción de los módulos probadores para cada caso específico, de forma tal que permite incluir flexiblemente los tipos de prueba que estime conveniente.
El Ambiente de operación de esta plataforma requiere de computadores interconectados en una red aislada. Debe haber un computador por cada alumno y un computador servidor de la plataforma para ser operado por el profesor.

El diagrama de la figura 1 describe la arquitectura de la plataforma.

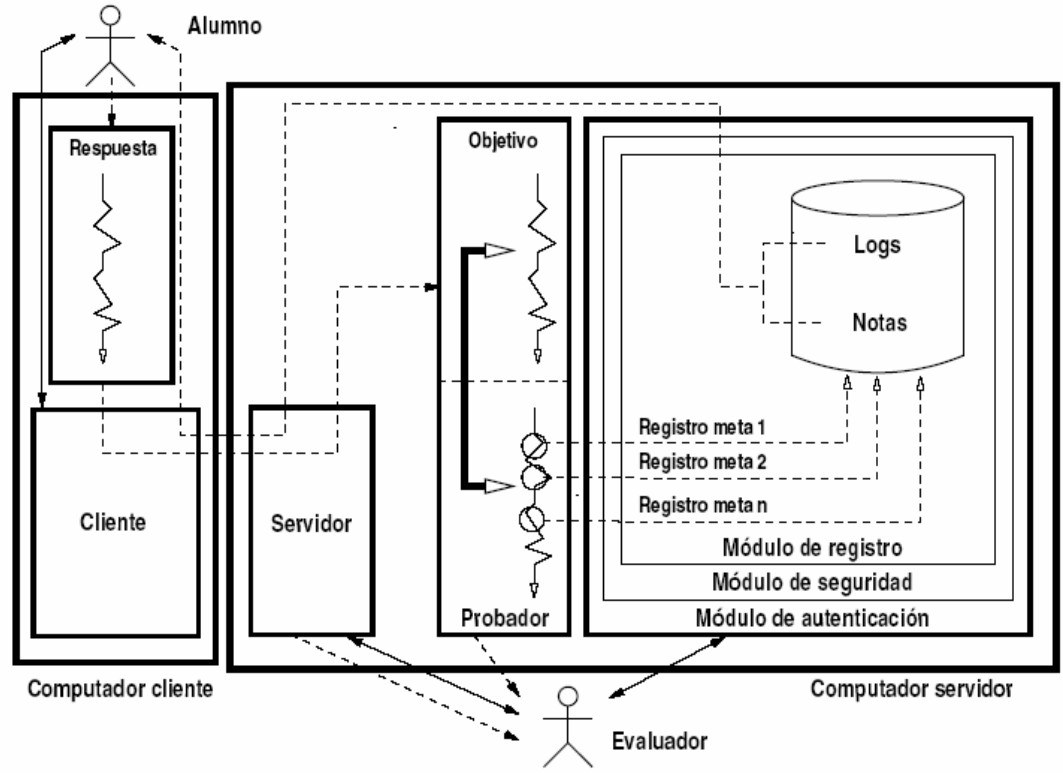

Figura 1. Arquitectura de la plataforma

La descripción de los módulos de la arquitectura representada en la figura 1 es la siguiente:

- Módulos permanentes.
  a) *Servidor*: recibe los archivos creados por los alumnos y activa los módulos de seguridad y luego, pone en marcha los módulos de prueba y Objetivo y envía un reporte al módulo cliente.
  b) *Cliente*: transmite los archivos creados por el alumno y la clave al módulo Servidor.
  c) *Registro*: módulos que registran las calificaciones cuando se alcanzan las metas de los módulos probadores.
  d) *Seguridad*: controla el acceso de los alumnos detectando si el alumno está enviando su respuesta desde el computador asignado.
  e) *Autentificación*: controla asociación la clave y los archivos enviados desde el cliente con directorios y archivos específicos de cada alumno.

- Módulos transitorios: diseñados por el profesor, pueden escribirse en un lenguaje distinto al de los módulos permanentes. Estos módulos pueden ser:
    a) *Probador de compilación y enlazado*: hace una evaluación del proceso de compilación y el enlazado de los componentes del módulo Objetivo.
    b) *Probador caso estático*: es un procesamiento sobre el código fuente del módulo Objetivo.
    c) *Probador caso dinámico*: interactúa en tiempo de ejecución con el módulo Objetivo.
    d) *Objetivo*: es el programa creado por el alumno como respuesta dentro del computador Servidor de la prueba.

5. Diseño del módulo Probador y casos de prueba.

***Caso A**: Curso de Lenguaje de Programación (ANSI C).*
Nuestro propósito era obtener módulos Probadores para programas relativamente pequeños, que el estudiante pudiera construir en un período de tiempo que no fuera superior a unas dos horas.
Nos propusimos especificar módulos Probadores que, además de realizar pruebas de tipo caja negra, permitieran evaluar la capacidad del estudiante para manejar las estructuras básicas del lenguaje, para usar listas encadenadas, arreglos, memoria dinámica, recursividad, para definir correctamente el paso de parámetros a funciones, etc.
Encontramos que el mecanismo apropiado para lograr que el módulo Probador tuviera acceso a estructuras del lenguaje, consistía en integrar el módulo Probador y el módulo Objetivo en un sólo programa ejecutable, interviniendo previamente, el código fuente creado por el estudiante. Esta intervención tiene por objetivo cambiar el nombre a la función *main* evitando la colisión con la función *main* del módulo Probador, además es para detectar la presencia de ciertos patrones cuya inclusión se exige o se prohíbe y, en ciertos casos, para agregarle código para, por ejemplo, determinar si una función se ejecuta o no en forma recursiva.
En los enunciados se pidió a los alumnos realizar la programación de cuerpos de funciones, las que, a objeto de permitir al módulo Probador ejecutar las diferentes llamadas de prueba, debían respetar prototipos específicos. Se les incluyó también requisitos tales como programar en forma recursiva o bien operar sobre listas encadenadas, que con esta estrategia de prueba se pudieron controlar efectivamente.

Las condiciones de las experiencias realizadas fueron las siguientes:
a) Aplicado a un curso de 32 alumnos.
b) La calificación tiene un peso de 50% del puntaje de la evaluación práctica, que incluye tareas individuales.
c) La duración de la prueba es de 1,5 horas.
d) Sistema operativo Linux.
e) Cada evaluación consiste en tres problemas de 2 puntos cada uno en la escala de 1 a 7.
f) Cada problema incluye dos metas.
g) Los tipos de prueba de programas son de caja negra, caja blanca a nivel de estructuras y funciones.
h) El alumno edita su programa respuesta respetando las restricciones de prototipos o nombres de variables, compilar y probar localmente y luego enviarlo a revisión por medio del módulo Cliente. El módulo Cliente le retroalimentaba con los resultados de la revisión y el puntaje obtenido (ver figura 2).

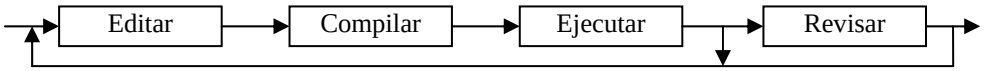

Figura 2: Pasos para responder.

Caso B: Curso de Redes, en tópico de Ingeniería de Protocolos.
En este caso, considerando que se está probando la capacidad del estudiante para realizar programas que se comuniquen en sobre una red siguiendo un protocolo dado como tarea, se permitió el reemplazado el módulo Servidor y el módulo Probador por un único programa servidor/evaluador que implementa una parte del protocolo, dejando al estudiante la tarea de completarlo mediante la creación de un programa que es la entidad par del servidor.
En la evaluación realizada al curso se generó un conjunto de metas que bebían ser alcanzadas por los programas de los alumnos, por ejemplo:
Establecer la conexión, transmitir la clave privada del estudiante, enviar correctamente un archivo al servidor/evaluador. El archivo enviado es otro programa fuente que el compilador compila y ejecuta para realizar una nueva comunicación, etc.
La retroalimentación se obtuvo realizando la consulta a la plataforma mediante un navegador.

Las condiciones de las experiencias realizadas fueron las siguientes:

a) Aplicado a un curso de 48 alumnos.
b) la calificación es 50% de la evaluación práctica (el resto son tareas en grupos de 3 alumnos)
c) se mide la capacidad de construir programas capaces de comunicarse mediante un protocolo específico.
d) Aspectos susceptibles de medición: eficacia, eficiencia, prestaciones, existencia de ciclos improductivos, existencia de bloqueos.
e) Aspectos medidos de tipo caja blanca con cobertura de trayectorias, prestaciones y existencia de bloqueos.

6. Análisis de resultados

En la figura 3 se muestra la distribución del puntaje obtenido por los alumnos de la asignatura Lenguajes de Programación de la carrera de Ingeniería Civil Informática (caso A), se pueden resumir:

a) La nota promedio fue 4,2 puntos (nota de aprobación: 4 en escala 1-7).
b) El 60% de los alumnos alcanzó 3 o más metas de 6 (calificación mayor o igual a 4).
c) El 3% de los alumnos no alcanzo ninguna meta.
d) El 11% de los alumnos alcanzo todas las metas.

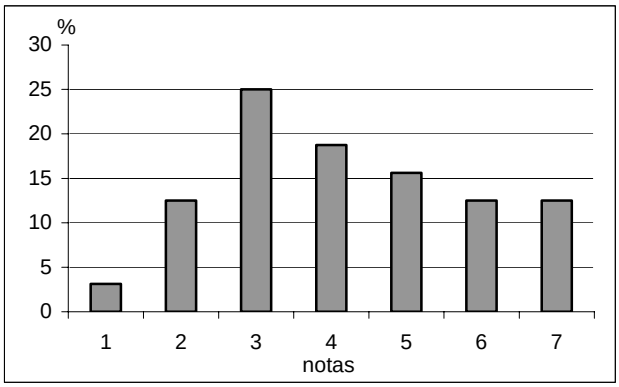

Figura 3 calificaciones caso A.

En la figura 4 se muestra la distribución del puntaje obtenido por los alumnos de la asignatura Redes de la carrera de Ingeniería Civil Informática (caso B), se pueden resumir:

a) La nota promedio fue 4,1 puntos (nota de aprobación: 4 en escala 1-7).
b) El 56% de los alumnos alcanzó 3 o más metas de 6 (calificación mayor o igual a 4).
c) El 8% de los alumnos no alcanzo ninguna meta.
d) El 15% de los alumnos alcanzo todas las metas.

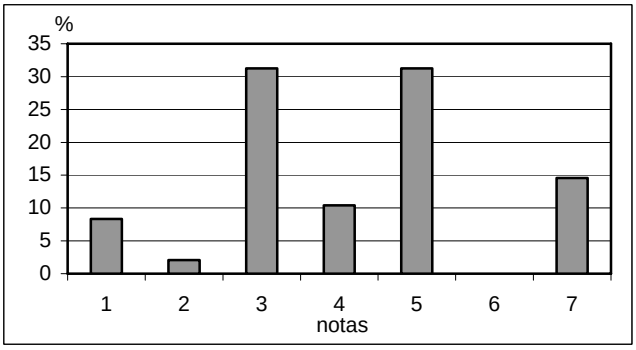

Figura 4 Calificaciones caso B.

El resultado obtenido en ambos cursos muestra una distribución de puntajes que no difiere significativamente de los casos de pruebas escritas. En ambos casos se observa que un bajo porcentaje de los alumnos no logró metas.

A Continuación se presenta una tabla que contiene el resumen de los resultados de la encuesta realizada a los estudiantes que fueron sometidos a la prueba con evaluación automática del curso de 48 alumnos del caso B.

	Pregunta	**Promedio (1-5)**
1	Esta fue una prueba justo de lo que aprendimos	4,1
2	Esta prueba fue fácil, si estudié para ello	4,1
3	Esta clase de prueba es nueva para mi	4,2
4	Esta clase de pruebas en verdad me hace pensar	4,0
5	Hice un buen trabajo de preparación para la prueba	3,9
6	El instructor hizo un buen trabajo para prepararnos para la prueba	3,5
7	Por desgracia me sorprendieron las preguntas elegidas	2,6
8	Las instrucciones fueron claras	4,0
9	El tipo de prueba me produjo un nerviosismo mayor que las de otro tipo	3,3
10	El obtener la calificación de inmediato resulta positivo	4,6
11	La plataforma de evaluación me parece fiable	4,2
12	La plataforma hace calificaciones justas	3,7
13	Este tipo de prueba mide habilidades que por otro medio no se puede	4,5
14	El número de preguntas es inadecuado para este tipo de pruebas	2,3
15	El tipo de problemas es adecuado para este tipo de pruebas	3,8
16	La interfaz y la retroalimentación de la plataforma es adecuada	3,9
17	Este tipo de prueba me pone en una situación similar a las de la vida laboral	3,5

Los alumnos debieron responder con una cruz en aquella opción que mejor representara su opinión. (Desde "completo desacuerdo" 1, hasta "completo acuerdo" 5).

Además de las preguntas de la tabla anterior, se les pidió que identificaran el aspecto de la evaluación automática que a su juicio es el más positivo y el más negativo, las respuestas más frecuentes fueron las siguientes:

- **Aspecto más positivo:** obtener la nota de inmediato y realmente pone a prueba la capacidad de programar.
- **Aspecto más negativo:** produce nerviosismo adicional por falta de experiencia en este tipo de evaluaciones.

De los resultados de la encuesta se desprende que los alumnos tienen una visión positiva de la evaluación automática. Sin embargo, queda claro que la retroalimentación de la plataforma es un aspecto a perfeccionar. Por otra parte, el nerviosismo que se produce en los estudiantes, aunque es un aspecto que amerita ser analizado en mayor profundidad, puede verse reducido en la medida que se utilice este tipo de evaluación con más frecuencia.
Considerando que en ambos casos de evaluación se hicieron tres problemas, que solamente se aplicó una única vez durante el curso y que la duración de la prueba fue de 1,5 horas, hemos llegado a la conclusión que sería mejor emplearla con una frecuencia semanal o quincenal a manera de test de entrada a sesiones de laboratorios. Eso si se debería limitar a la solución de un sólo problema dividido entre 2 y 4 metas, con una duración máxima 0,5 horas. La principal dificultad consiste en disponer de la infraestructura de laboratorio adecuada para un curso completo y de una batería amplia de problemas adecuados a todas las etapas del curso.
El tipo de problemas que se pueden incluir en este tipo de pruebas, considerando lo limitado del tiempo, no puede ser de gran complejidad por lo que resulta recomendable sólo en las instancias básicas de la programación.

7. Conclusiones

El proceso de evaluación automática, descrito en este artículo, ahorra considerable tiempo y esfuerzo a los docentes e instructores de los cursos que incluyen programación. El tiempo y esfuerzo ahorrados son mejor empleados por el docente en la construcción de especificaciones sólidas y claras tanto para el módulo Probador como para el estudiante. El mecanismo de evaluación utilizado por esta plataforma da garantías de imparcialidad, dado que proporciona calificaciones automáticamente en base al cumplimiento de requisitos específicos.
La complejidad de los programas a evaluar no está limitada por la plataforma automática, sino más bien por los objetivos específicos que se desean calificar. Por ejemplo, para el caso de pruebas frente a un computador la complejidad de los programas no puede ser tan alta principalmente por las limitaciones de tiempo. Sin embargo, la complejidad de las tareas de programación puede ser mayor dado que los estudiantes disponen de más tiempo y su

limitación radica en la factibilidad de construir módulos Probadores que cumplan los objetivos a evaluar por el profesor.
La modalidad de evaluación automática ha mostrado ser un complemento efectivo a las otras formas de evaluación y permite mejorar la metodología de enseñanza y evaluación entregando al instructor una retroalimentación oportuna del estado general del curso.
De acuerdo a las experiencias realizadas consideramos que la plataforma automática presentada en este artículo no sustituye los mecanismos tradicionales de evaluación, pero hemos probado que los complementa eficazmente, supliendo algunas de sus deficiencias.
En cada instancia de evaluación se genera documentación permanente de cada programa que se prueba y de los módulos Probadores. Estos últimos pueden ser utilizados tanto para nuevas pruebas como para ser adaptados a plataformas de enseñanza no presencial.
Por otra parte, se ha podido comprobar que los estudiantes aprecian que se les enfrente a esta forma de evaluación, porque consideran que los pone realmente a prueba y los capacita para crear programas. Además los resultados muestran que las calificaciones obtenidas por los estudiantes utilizando la plataforma automática se encuadran en los rangos normales de calificaciones obtenidos por los alumnos usando la metodología tradicional de evaluación.
Finalmente, podemos afirmar que el profesor tiene una herramienta que le garantiza que los alumnos que aprueban estas evaluaciones, han logrado la capacidad de programar a cierto nivel.

8. Trabajo futuro

Nos proponemos investigar los aspectos y mecanismos de prueba de programas con interfaz gráfica y con lenguajes orientados al objeto. Además trabajaremos en una plataforma con un repositorio para revisión automática de tareas de programación y profundizaremos la prueba estática de programas para abordar la construcción de un asistente de programación con retroalimentación que incluya un diagnóstico más completo del estado del estudiante. Finalmente, investigaremos la generación automática de Probadores para evaluar objetivos de acuerdo especificación de requisitos.

REFERENCIAS

[1] Boehm B. W. “Software Engineering: R & D Trends and Defense Needs”. En Research Directions in Software Technology (Wegner P., ed.) Cambridge MA: MIT Press, 1979.
[2] Canup M. J. and Shacketford, R. L, “Using Software to Solve Problems in Large Computing Courses”, “Proceedings SOGCSE’98, Atlanta, GA, USA, 135-139.
[3] Jones, E. L., “The SPRAE Framework for Teaching Software Testing in Undergraduate Curriculum”, Proceedings ADMI 2000, June 1-4, 2000,Hampton, VA USA.
[4] Jones E. L and Allen C. S., “Grading Student Programs- A Software Testing Approach”, Journal of Computing in Small Colleges,16, 2, January 2001, 185-192.
[5] Millar, E., “The Philosophy of Testing” En Program Testing Techniques, IEEE Computer Society Press, 1977, pp, 1-3.
[6] Myers G. J. “The Art of Software Testing”. New York: John Wiley & Sons. 1979.
[7] Zhu, H., Hall, P.A.V., and May, J.H.R., “Software Unit Test Coverage and Adequacy”, ACM Computing Surveys, Vol. 29, No. 4, December 1997, pp. 366-427.
[8] Martínez Arias, Rosario, “Teoría de los Test Sicológicos y Educativos”. Cáp. 13, La validez Relativa al criterio: Test predictor y un criterio. Editorial Síntesis SA. Madrid, España, 1996.