AUTO EFICACIA Y EVALUACIÓN AUTOMÁTICA

Jorge López Reguera, Cecilia Hernández Rivas, Yussef Farran Leiva

RESUMEN

En este artículo se describe una plataforma de prueba automática de programas facilita el aprendizaje de los estudiantes de cursos de programación y apoya a los docentes entregándoles un nuevo instrumento de evaluación con alumnos frente al computador. La plataforma también permite elaborar estrategias orientadas a reforzar la *auto eficacia* en los alumnos y a posibilitar la construcción de *modelos mentales* viables [8]. Se presenta la experiencia obtenida en las asignaturas de *lenguaje de programación s* en tres semestres. 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 en cursos que incluyan programación.

Palabras claves: e-learning, modelos mentales, auto eficacia, evaluación automática, prueba de programas.

[1]Depto de Ingeniería Informática y Ciencias de Computación, Facultad de Ingeniería - Universidad de Concepción
jorlopez@udec.cl *cecihernandez@udec.cl* *yfarran@udec.cl*

INTRODUCCIÓN

Algunos estudios sugieren que la tasa de fracaso en un primer curso de programación oscila en torno a un 30 por ciento [6]. Los estudiantes que están en ese porcentaje suelen creer que no podrán aprender a programar o que no serán eficientes en programación por el resto de su vida profesional. Aún dentro del porcentaje de los alumnos que aprueban las asignaturas de programación se observa que no hay un nivel homogéneo de conocimiento ni se tiene la garantía de que los sistemas de evaluación detectan el nivel mínimo requerido. Por otra parte, los sistemas de evaluación tradicionales no siempre cumplen con requisitos de oportunidad que permita detectar a tiempo a los alumnos débiles para identificar en qué consiste esa debilidad y realizar las acciones tendientes a superarla.

Aprender a programar es una experiencia única para cada estudiante, y no es totalmente conocido por qué una persona en un curso introductorio aprende a programar más rápidamente que otra. En la literatura encontramos varios factores que pueden influir en el éxito de los novatos en programación tales como experiencia previa en computación [14, 15], comodidad mientras se trabaja [16], sensación de estar jugando durante el entrenamiento [16, 17], ***auto eficacia*** [8, 18], conocimientos previos en matemáticas y ciencias [16], estilo de aprendizaje [19], y el ***modelo mental*** que el estudiante tiene de la programación [8, 20, 21].

La auto eficacia definida por Bandura [1] es "un juicio que una persona realiza de su propia habilidad para organizar y ejecutar cursos de acción que le permitan alcanzar un cierto nivel de desempeño en un cierto dominio". Influye en la cantidad de esfuerzo empleado, persistencia en el caso de fallar y en el rendimiento logrado. Postula que en el aprendizaje, los juicios de auto eficacia se fundan en cuatro fuentes de información: el rendimiento en los logros personales logros previos, la observación de que sus semejantes pueden hacerlo, la persuasión verbal y el estado psicológico desde el cual el alumno juzga sus capacidades.

Los resultados obtenidos en Vennila [8], para el caso específico de la programación, muestran que la *auto eficacia* es afectada significativamente al desarrollar *modelos mentales* adecuados. Además, concluye que tanto lo que los alumnos saben, representados por sus *modelos mentales*, como lo que ellos creen de si mismos, representados por la *auto eficacia*, impactan directamente en su rendimiento en un curso de programación. Vennila recomienda que para favorecer el desarrollo de una buena auto eficacia, es necesario que el ambiente de aprendizaje permita la medición de superación individual, la solución de problemas de complejidad incremental, la búsqueda de soluciones en equipo, el desarrollo de *modelos mentales* adecuados y la asignación de tareas frecuentes y cortas.

Los *modelos mentales*, de acuerdo a Götshi [9], son las representaciones cognitivas que un estudiante construye en la intimidad de su mente. Por otra parte, el *modelo conceptual* es la herramienta que el profesor usa para enseñar conceptos, y constituye la versión "comunicable" del *modelo mental*. Kahney [10] encontró que los estudiantes novatos poseen *modelos mentales* que son una variante de los modelos conceptuales y no es raro que algunos resulten *no-viables*.
Particularmente, en la educación de ciencias de computación, se han usado *modelos mentales* para describir el conocimiento de los estudiantes, como es el caso del estudio realizado por Vennila [8], en el que se encontró que si bien la mayoría de los alumnos desarrollaron el modelo mental de *copias*, considerado "viable" para la recursividad, muchos otros construyeron modelos considerados "no viables" tales como *ciclo, activo, paso, mágico y de valor de retorno*. La metodología que pusieron en práctica, consistió en obtener desde pruebas

y exámenes manuscritos una clasificación realizada en forma visual de los signos y diagramas que los alumnos usaron al hacer el seguimiento de los algoritmos planteados como problema.

Sin duda, conocer el *modelo mental* que cada alumno posee, mejora sustancialmente las posibilidades del proceso de enseñanza-aprendizaje al permitir abordar los casos particulares de cada alumno en forma diferenciada, pero significa un desafío de esfuerzo adicional a los docentes que se incorpora tanto al proceso de entrenamiento como al de evaluación en programación. Estos cursos de programación, muchas veces numerosos, imponen una demanda adicional en comparación con los cursos tradicionales que están diseñados para soportar principalmente evaluaciones escritas [2]. La metodología tradicional de evaluación en cursos de programación, incluye:

a) *Pruebas escritas*.
b) *Tareas de programación*.
c) *Programación frente al computador*.

Las principales deficiencias de la metodología descrita arriba incluyen:

1. Imposibilidad de realizar controles frecuentes, en base a objetivos específicos, en cursos numerosos.
2. Dificultad para realizar evaluaciones significativas individuales del proceso de aprendizaje. Ej.: *modelos mentales, estilos de aprendizaje, nivel de auto eficacia, etc.*.
3. Posibilidad de plagio de tareas de programación.
4. Dificultad para evaluar imparcialmente.

Las deficiencias de la metodología de enseñanza 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 aprendizaje reforzando las posibilidades de evaluación (tipo c), con el propósito de lograr los siguientes objetivos:

1. Someter al alumno a una situación similar a lo que experimentará en su actividad profesional.
2. Obtener información más profunda y significativa de la situación general del curso y particular de cada alumno que incluya *auto eficacia* y *modelos mentales*.
3. Medir con una granularidad más fina para que los alumnos y profesores puedan ajustar su esfuerzo y reforzar los aspectos deficitarios a tiempo.
4. Garantizar que cada alumno sea capaz de construir programas.
5. Entregar una calificación imparcial, a cada alumno, al finalizar el curso.

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 programas”.

Este artículo está organizado como sigue: la sección 2 revisa los fundamentos de la prueba de programas, la sección 3 describe la arquitectura de la plataforma, la sección 4 describe la metodología usada en tres semestres de aplicación, la sección 5 muestra el análisis de los resultados de la aplicación de la plataforma. Finalmente la sección 6 presenta conclusiones y el trabajo futuro.

PRUEBA DE PROGRAMAS Y TRABAJOS RELACIONADOS

Según Millar [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”. Boehm [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 Zhu [7], se definen los siguientes criterios para la Adecuación de la Prueba de Programas: cobertura de sentencias, cobertura de transferencias de control, cobertura de trayectorias y adecuación de mutaciones.

Los objetivos de la prueba de programa, según se desprende de lo afirmado por Dijkstra en 1972 y de las reglas establecidas por Myers [4, 6], sólo pueden demostrar la existencia de un conjunto de defectos específicos en el software, pero no la ausencia total de defectos.

Existen varias plataformas de revisión automática de programas, la más conocida es ACM Programming Contest [23] (competencia de programación), cuyo objetivo es encontrar el equipo ganador de un campeonato de internacional de programación para estudiantes. La plataforma obtiene un ranking de los equipos participantes seleccionando como ganador al que resuelve correctamente más problemas en la menor cantidad de tiempo. La plataforma hace un análisis estático del tipo "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 Jones [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 [13], 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.

En [22], se presenta una estrategia de análisis estático de programas en lenguaje en Java. El propósito de este trabajo es una plataforma para mejorar la calidad de la programación, ofreciendo a sus usuarios un conjunto de recomendaciones tales como: minimizar el número de variables, evitar el obscurecimiento de de variables, simplificar la complejidad de la estructura del programa, etc.

Nuestro trabajo tiene como meta principal que los estudiantes desarrollen *modelos mentales viables* y eleven su *auto eficacia*. Para ello hemos abordado el problema desde la perspectiva de la evaluación, basándonos en el enfoque actual de la educación: *no existen alumnos estándar, por lo tanto la enseñanza y la evaluación deben ser personalizadas y variadas, poniendo énfasis en el aprendizaje.*

En el diagrama de la Figura 1, representa el modelo de apoyo al aprendizaje que hemos usado para diseñar la plataforma las flechas segmentadas representan efecto, las líneas continuas representan flujo de información.

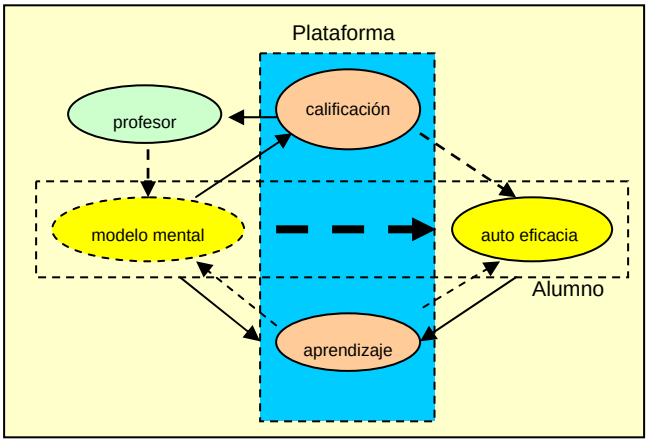

Figura 1: Modelo de apoyo al aprendizaje.

La plataforma tiene los siguientes objetivos específicos:

a) Realizar calificaciones automáticas con entrega inmediata de resultados.
b) Apoyar el aprendizaje en base a resolución de problemas, en sesiones de laboratorio evaluadas *significativamente*, para retroalimentar de la forma más completa y oportuna posible, tanto a alumnos como instructores.
c) Motivar el autoaprendizaje entregando retroalimentación y sugerencias en forma automática.

ARQUITECTURA DE LA PLATAFORMA

Para lograr los objetivos específicos la plataforma ofrece dos tipos de ambiente: uno consistente en una red cerrada conectada a un *Centro de Evaluación* para el proceso de calificación y un ambiente con acceso web que también incluye al *Centro de Evaluación,* para apoyar el aprendizaje (ver Figura 2).

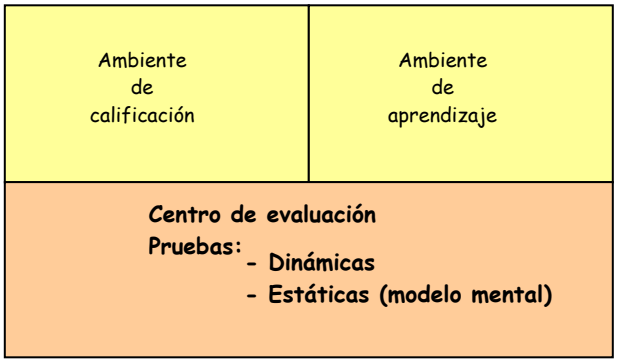

Figura 2:.Arquitectura general de la plataforma

El **ambiente de calificación** está orientado 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. Ofrece una retroalimentación mínima que incluye las calificaciones parciales durante toda la prueba y mensajes de error básico.

El **ambiente de aprendizaje**, puede ser usado tanto para auto aprendizaje (actualmente en construcción) como para sesiones en las que usa la metodología de aprendizaje por resolución de problemas, con presencia de instructores. En este ambiente el alumno recibe la retroalimentación máxima, que incluye identificación del modelo mental y sugerencias.

Como se muestra en la Figura 3, el alumno puede editar, compilar y ejecutar localmente su programa respuesta y enviarlo al centro de evaluación, tantas veces como quiera. El centro evaluador lo retroalimenta con los resultados de la revisión y el puntaje obtenido. El centro evaluador mantiene el mejor puntaje obtenido.

Centro
Evaluador

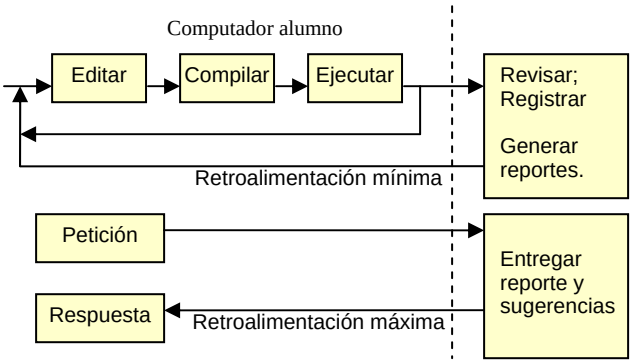

Figura 3: Pasos para responder.

El **Centro de Evaluación** procesa los programas respuesta de los alumnos de acuerdo al esquema de la Figura 4. Los somete tanto a pruebas dinámicas como estáticas. Las pruebas estáticas realizan la verificación del cumplimiento de metas de ejecución de los programas. La prueba estática tiene el propósito de apoyar la identificación del modelo mental mediante la extracción de muestras significativas del programa fuente. Toda la actividad que el alumno realiza queda almacenada en el registro histórico. La evaluación consiste en determinar cual es el *modelo mental* más probable, a partir del resultado de la prueba dinámica (caja negra y caja blanca), de la clasificación del perfil resultante de la prueba estática y (en el futuro) de la historia de los intentos previos.

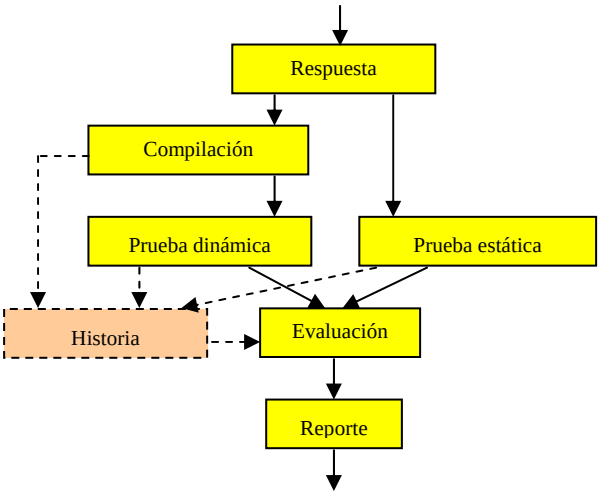

Figura 4: .Evaluación automática de programas

El diseño de la plataforma es modular y está basado en el modelo de comunicación *cliente-servidor*.

El Ambiente de calificación 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 que constituye el *Centro de Evaluación* es operado por el profesor.

Interfaz del alumno:

$INSC jlr123
.....
$TEST r1.c jlr123 1
Meta 1.a: timeout!
Meta 1.b: violacion de segmento!
Meta 1.c: ok!
|bas|1.a|1.b|1.c|2.a|2.b|2.c||Tot|

|1.0|0.0|0.0|1.0|0.0|0.0|0.0||2.0| jlr123

Interfaz del Centro de Evaluación:

IP: 152.74.52.17, alumno: jlr123 pregunta: 1
****
****
Compilacion: ok!
.....
Ejecucion:
Meta 1.a: tiemout!
Meta 1.b: violacion de segmento!
Meta 1.c: lograda!
.....
|bas|1.a|1.b|1.c|2.a|2.b|2.c||Tot|

|1.0|1.0|1.0|0.0|0.0|0.0|0.0||3.0|
alumno1
|1.0|1.0|1.0|1.0|0.0|0.0|0.0||4.0|
alumno2
|1.0|1.0|1.0|1.0|0.0|0.0|0.0||4.0| .....

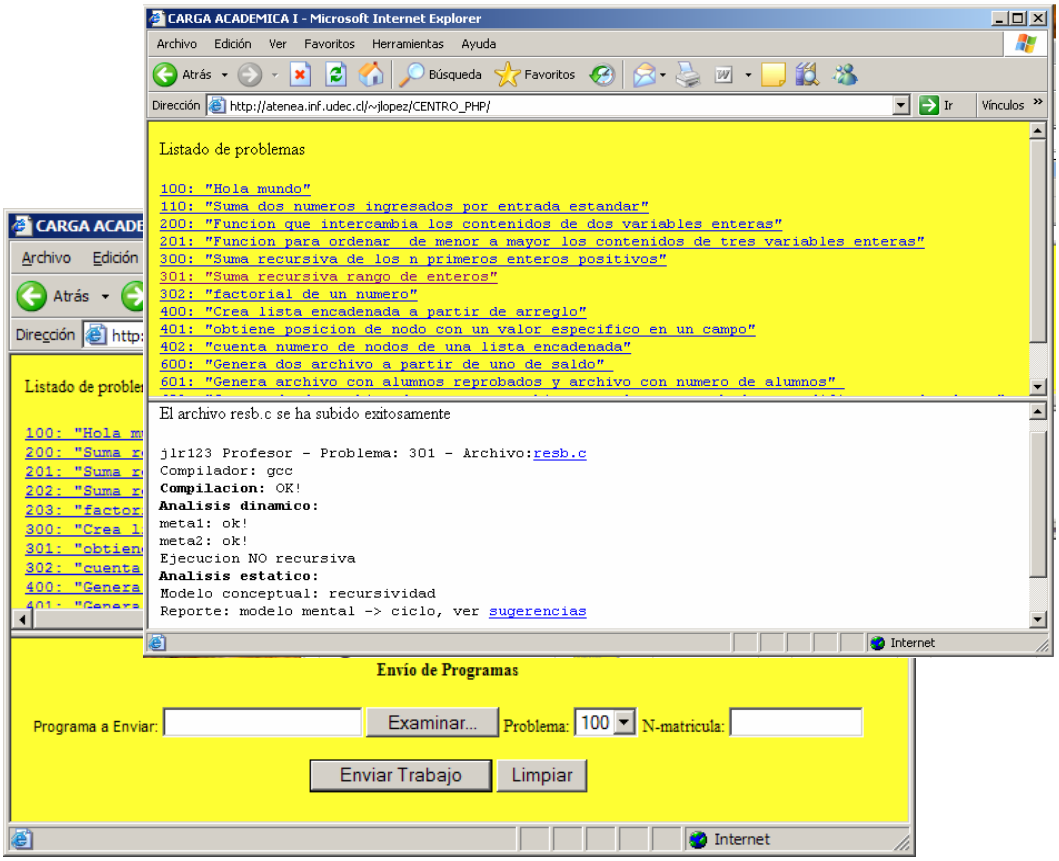

Figura 5: Interfaz del ambiente de aprendizaje

METODOLOGÍA

La plataforma se ha ido usando en la medida que lo ha permitido la incorporación de nuevos módulos. El semestre 2004-1, sólo se usa el ambiente de calificación (primera herramienta de apoyo a la docencia que estuvo disponible). El semestre 2005-1, los alumnos practicar con la versión más básica del ambiente de aprendizaje, que consiste en una versión web del ambiente de calificación. El semestre 2006-1 se usa el ambiente de aprendizaje con capacidades básicas de detección de algunos modelos mentales encontrados en la literatura [8] desde el comienzo del semestre en sesiones piloto de laboratorio y en forma libre por los alumnos.

Las pruebas en el ambiente de calificación en los tres semestres son similares, con la misma cantidad de problemas y metas por problema y su aplicación es en el primer curso de programación de la especialidad. Este curso es en *Lenguaje de Programación (ANSI C y C++).*

Los problemas están orientados a programas relativamente pequeños, de modo que el estudiante pueda construirlos en un período de tiempo que no superior a dos horas.

En los enunciados se pide a los alumnos realizar la programación de cuerpos de

[1]Depto de Ingeniería Informática y Ciencias de Computación, Facultad de Ingeniería - Universidad de Concepción
jorlopez@udec.cl cecihernandez@udec.cl yfarran@udec.cl

funciones, las que, a objeto de permitir al módulo Probador ejecutar las diferentes llamadas de prueba, deben respetar prototipos específicos. Se les incluye también requisitos tales como programar en forma recursiva o bien operar sobre listas encadenadas, que con esta estrategia de prueba se pueden controlar efectivamente.

Nuestra experiencia incluye la aplicación de la evaluación automática durante tres semestres a cursos de unos 40 alumnos:

a) 2004-1: sólo se usó el ambiente de evaluación sin que los alumnos hubieran practicado con él, una vez al final del semestre.
b) 2005-1: se usó la primera versión del ambiente de aprendizaje la semana anterior a la evaluación del final del semestre.
c) 2006-1: durante todo el semestre se usó una versión del ambiente de aprendizaje que incluye detección de modelos mentales y calificación del proceso de aprendizaje.
d) Duración de la evaluación es de 1,5 horas.
e) Cada evaluación consiste en tres problemas de dos o tres metas.

Ejemplo problema a evaluar:

- **Función***: buscar en una lista encadenada la posición del nodo que contiene el valor k almacenado en el campo x. La función debe tener exactamente el prototipo:*

```
int numeroNodo(struct nodo *, int);
  struct nodo {
        int x;
        struct nodo *p;};
```

- **Argumentos***:*
  - *Puntero al último elemento de una lista.*
  - *Entero k.*
- **Retorno***:*
  - *Entero n:{1,2,..}, con la posición del nodo con entero k.*
  - *0, si k no se encuentra en la lista.*

ANÁLISIS DE RESULTADOS

La Figura 6 muestra la distribución de los puntajes obtenidos en las calificaciones realizadas durante los tres semestres. En este gráfico se puede apreciar que los puntajes tienden a mejorar en los semestres sucesivos, como también se puede apreciar en el gráfico de la Figura 7, en el que se muestra el alcance de metas. Se puede ver en estos dos gráficos de la Figura 6 y 7, que en el semestre 2005-1 hay un aumento en el número de alumnos que alcanzan todas las metas y por lo tanto la nota máxima respecto del semestre 2004-1, esto es mucho más significativo al comparar el semestre 2006-1 con el semestre 2005-1.

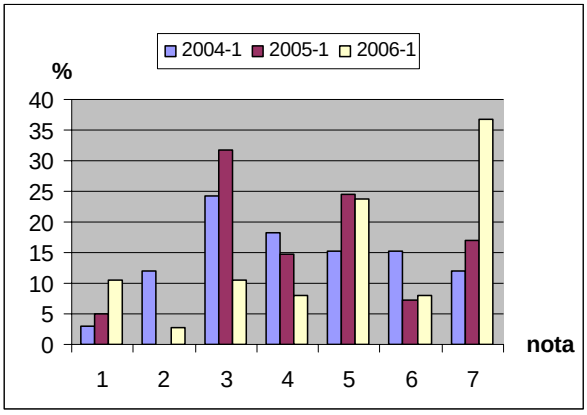

Figura 6:: Distribución de calificaciones (Rango de notas: 1-7)

También se puede apreciar que el porcentaje de alumnos aprobados muestra la misma tendencia a aumentar pero es más significativa al comparar los últimos dos semestres. Los alumnos que obtienen la nota mínima (ninguna meta), aunque en porcentaje aumentan, en realidad el máximo

es de cuatro alumnos, lo cual está dentro de los rangos normales.

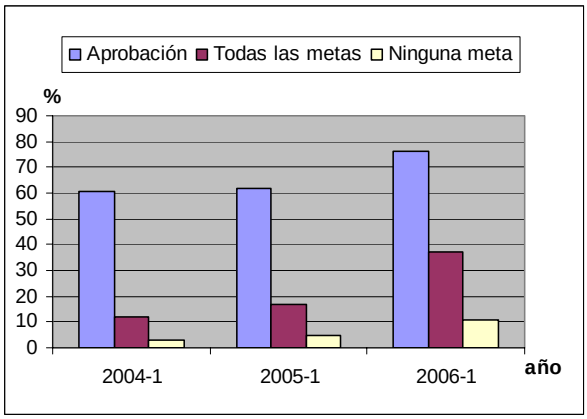

Figura 7: Alcance de metas. (nota de aprobación: 4)

El gráfico de la Figura 8 también nos muestra una tendencia positiva respecto de los promedios de notas del curso, especialmente en el semestre 2006-1.

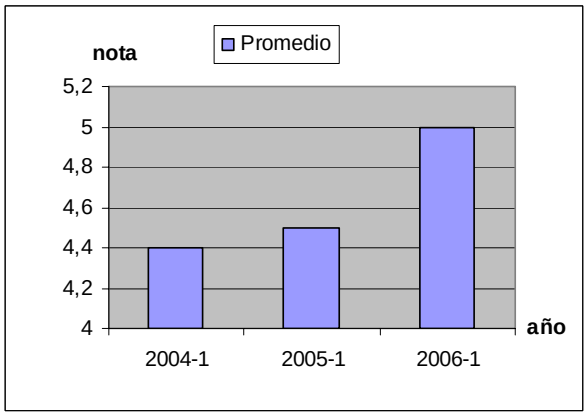

Figura 8: Evolución del promedio (Rango de notas: 1-7)

Observando los resultados obtenidos en los tres semestres, se nota una cierta correlación entre el nivel de experiencia previa la calificación que los alumnos tuvieron en cada semestre. El semestre 2006-1, ellos pudieron practicar libremente y también en sesiones de aprendizaje en laboratorio.

Adicionalmente, los estudiantes respondieron una encuesta de 15 preguntas, para ver la percepción que ellos tienen de la plataforma. Los estudiantes debieron elegir aquella opción que mejor representara su opinión. (Desde "completo desacuerdo" 1, hasta "completo acuerdo" 5), por ejemplo: obtener la calificación de inmediato me resulta positivo, la plataforma hace calificaciones justas, este tipo de pruebas mide habilidades que por otro medio no se puede, el tipo de problemas es adecuado a este tipo de pruebas, etc.

De los resultados de la encuesta se desprende que los alumnos tienen una visión positiva de la evaluación automática. La mayoría de los alumnos consideran lo más positivo de la plataforma es *obtener la nota de inmediato* y realmente pone a prueba la capacidad de programar. También coinciden en lo más negativo es el nerviosismo adicional por falta de experiencia en este tipo de evaluaciones.

El semestre 2006-1 se pudo percibir disminución del tiempo promedio al alcanzar las metas, lo que estimamos como un aumento en la seguridad en los alumnos, sin embargo para sacar conclusiones definitivas al respecto, se deben obtener más datos que permitan un análisis más profundo.

El semestre 2006-1, durante las sesiones de aprendizaje en laboratorio, se pudo detectar varios casos de modelos mentales no viables que fueron atendidos por los instructores.

CONCLUSIONES Y TRABAJO FUTURO

El ambiente de aprendizaje ha mostrado tener un efecto positivo en el desempeño de los estudiantes, a pesar que aún se encuentra en sus primeras fases de desarrollo. A partir de los resultados obtenidos en las mediciones, se puede concluir que existe una correlación entre el nivel de ejercitación con el ambiente de aprendizaje y el desempeño logrado en el ambiente de calificación, pero creemos que hace falta profundizar el estudio para obtener conclusiones más específicas.

Para el caso de pruebas frente a un computador la complejidad de los programas no puede ser muy alta principalmente por las limitaciones de tiempo. Sin embargo, la complejidad de las tareas de programación en el ambiente de aprendizaje puede ser mayor, puesto que los estudiantes pueden disponer de más tiempo.

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.

El ambiente de calificación de esta plataforma constituye un *instrumento de evaluación* pedagógica, pero queda en manos de cada profesor darle un uso adecuado, validando las pruebas y los problemas que se aplican en cada oportunidad.

Finalmente podemos resaltar 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.

Trabajo futuro

El ambiente de calificación automática se ha usado ya en varias oportunidades permitiendo depurar la plataforma en la prueba dinámica de programas. Dentro de este proyecto nos proponemos profundizar la investigación acerca de la prueba estática de programas para aportar mayor información sobre los modelos mentales.

En el contexto de aprendizaje por resolución de problemas, queda pendiente el desafío de ampliar la plataforma para evaluar el proceso de grupos de estudiantes, de forma que se pueda aportar información más significativa respecto de las competencias.

Nuestro futuro trabajo contempla mejorar la captura de información relacionada con el comportamiento de los alumnos durante el proceso de aprendizaje, de forma que sea posible encontrar nuevos modelos mentales para poder discernir, incluso, estilos de aprendizaje.

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] Vennila, R., Labell,e D., Wiendenbeck, S., “Self-Efficacy and Mental Models in Learning to Program”, ACM SGCSE Bulletin, Vol. 36, Issue 3, June 2004.

[9] Götshi, T., Sanders, I., Galpin, V., “Mental Models of Recursion”, ACM

SGCSE Bulletin, Vol. 35, Issue 1, January 2003.

[10] Kahney, K., Sanders I., Galpin V., "What do a novice programmers Know about recursion?", En Studing de Novice Programmer, E.Soloway and J. Spohrer, Eds. L. Erlbaum, Hillsdale, New Jersey, 1989, pp. 315-323.

[11] Guzdial, M., Soloway E., "Log on education: teaching the Nintendo generation to program", Communication of the ACM, 45 issue 4, April 2002.

[12] Bandura, A., "Social Fundation of Thought and Action", Prentice Hall, Englewood Cliffs, NJ, 1986.

[13] Reek, K. A., "The TRY System - or – How to Avoid Testing Students Programs", Proceedings SIGSEC Bulletin vol. 21, no. 1, February 1989. PP. 112-116.

[14] Byrne, P. & Lyons, G. The effect of student attributes on success in programming. Proceedings of ITiCSE 2001, ACM Press, NY, págs. 49-54. 2001.

[15] Taylor, H. & Mountfield, L. Exploration od the Relationship between prior computing experience and gender on success in college computer science. Journal od Educatinal Computer Research, 11, págs. 291-306. 1994.

[16] Potosky, D. A field study of computer efficacy beliefs as an outcome of training: the role of computer playfulness, computer knowledge, and performance during training. Computer in Human Behavior, 18, págs. 241-255. 2002.

[17] Wilson, B. C. & Shrock S. Contributing to succes in an introductory computer çscience course: study of twelve factors. Proceedings of SIGCSE 2001, ACM Press, NY, págs. 184-188. 2001.

[18] Karsten R. & Roth R. M. Computer self-efficacy: a practical indicator of student computer comptency in introductory IS courses. Informing Science. 1(3), págs. 61-68. 1998.

[19] Thomas. L. Woodbury, J. & Jarman, E. learning styles and performance in the introductory programming sequence. Proceedings of SIGCSE 2002. ACM Press, NY, págs. 33-37.

[20] Cañas, J. J., Bajo, M. T. & Gonzalvo, P. Mental models and computer programming. Intertional Journal of Human Computer Studies, 40 (5), págs. 795-811. 1994.

[21] Soloway E., Ehrlich K. Empirical studies of programmer knowledge. IEEE Trnasaction of Sostware Egineering, SE 10 (5), pásg. 595-609. 1984.

[22] Nghi Truong, Paul Roe, Peter Bacroft. "Atatic Alaysis od Students' Java Programs". Sixth Australian Computing Education Conference (ACE2004), Dunedin, New Zealand Conferences. Conferences in Research and Practice in Information Technology., Vol. 30. 2004.

[23] International ACM Programming Contest. http://icpc.baylor.edu/icpc/. Acceso Julio de 2006.