2009

Primer curso de Computación para Ingeniería – Descripcion y ensenanzas de una experiencia practica

Marcelo Mejia
ITAM, marcelo.mejia@itam.mx

Alejandra Barrera
ITAM, abarrera@itam.mx

Silvia Guardati
ITAM, guardati@itam.mx

Primer curso de Computación para Ingeniería – Descripción y enseñanzas de una experiencia práctica

:

Marcelo Mejía
Instituto Tecnológico Autónomo de México
marcelo.mejia@itam.mx

Alejandra Barrera
Instituto Tecnológico Autónomo de México
abarrera@itam.mx

Silvia Guardati
Instituto Tecnológico Autónomo de México
guardati@itam.mx

RESUMEN

El uso de la estrategia primero-objetos en el curso introductorio de Computación para programas de Ingeniería permite que los alumnos desarrollen aplicaciones interactivas interesantes al término de un semestre. Esta estrategia se facilita al introducir los conceptos fundamentales de la programación orientada a objetos usando Alice y al hacer referencia continua a los conceptos aprendidos de manera visual al momento de desarrollar aplicaciones en Java. Para que el curso cubra el objetivo tradicional de desarrollar en el alumno la capacidad de analizar y resolver problemas de forma metódica y de expresar las soluciones de los mismos en términos de algoritmos, se hace énfasis en el uso adecuado y eficiente de las estructuras de control de flujo clásicas en la programación de los métodos de las clases.

Palabras clave

Enseñanza, Computación, Ingeniería, experiencia práctica.

INTRODUCCIÓN

El Departamento Académico de Computación del Instituto Tecnológico Autónomo de México (ITAM) ofrece, en primer semestre, un curso introductorio de Computación a los alumnos de los cuatro programas de Ingeniería del Instituto: Computación, Telemática, Industrial y Negocios. Históricamente, este curso de *Algoritmos y Programas* se ha centrado en el estudio a profundidad de las estructuras algorítmicas de control de flujo clásicas, selectivas y repetitivas, su aplicación a la solución de problemas complejos pero pequeños y su implementación en un lenguaje procedural (inicialmente Pascal y desde hace varios años C). Este enfoque de programación resulta interesante para los alumnos de Ingeniería en Computación, pero demasiado árido para los demás estudiantes.

Con el propósito de mejorar la percepción de la utilidad del curso entre todos los alumnos de Ingeniería y conciliar los intereses de los cuatro planes de estudio, se realizó una revisión de la literatura para conocer las diferentes alternativas que se están empleando actualmente para enseñar a programar en distintas partes del mundo. A partir de esta revisión y de los objetivos generales de los programas de Ingeniería del ITAM, se decidió modificar el enfoque del curso.

En las siguientes secciones se presentan, en orden, el objetivo del artículo, una revisión de la literatura relevante, el diseño del nuevo curso, algunos de los resultados obtenidos de su implementación, y las conclusiones que se obtuvieron.

OBJETIVO

El objetivo del presente trabajo es describir la estructura de nuestro nuevo primer curso de Computación para Ingeniería y presentar algunas de las lecciones aprendidas al impartirlo por primera vez, de tal manera que la experiencia pueda ser útil para otros programas de Ingeniería o de Sistemas o Tecnologías de Información en Latinoamérica.

El curso se enfoca ahora en el modelado ingenieril de problemas no triviales utilizando objetos y en su implementación mediante aplicaciones Java interactivas. El curso utiliza la estrategia de enseñanza primero-objetos y cuida la formación del pensamiento algorítmico en el alumno, que representa una habilidad muy buscada en todos los profesionales (Wing, 2006).

ESTRATEGIA PRIMERO-OBJETOS

Esta sección describe antecedentes que promovieron la enseñanza temprana de los objetos en los cursos introductorios de programación en carreras de computación, experiencias diversas de enseñanza del enfoque primero-objetos, así como herramientas que soportan el aprendizaje del mismo.

Recomendaciones de la ACM

La ACM ha formulado recomendaciones sobre el plan de estudios de la carrera de Ciencias de la Computación en varias ocasiones desde 1965, y aunque se trata de un campo dinámico, ha existido consenso desde entonces en cuanto a sus fundamentos: el rol de las matemáticas, y la importancia de enseñar a programar en los cursos introductorios. La guía propuesta por la ACM en 1968 enfatizó el pensamiento algorítmico en el primer curso, afirmando que la noción de algoritmo debía distinguirse claramente del concepto de programa. En contraste, el reporte de 1978 no mencionó el concepto de algoritmo, sino que resaltó la importancia de "estudiar lenguajes de programación para implementar estructuras de datos en el ambiente del hardware" (Hemmendinger, 2007). En 1991 se rescató nuevamente la noción de algoritmo afirmando que "la disciplina de la computación es el estudio sistemático de procesos algorítmicos que describen y transforman información: sus teorías, análisis, diseño, eficiencia, implementación, y aplicación" (Hemmendinger, 2007). El reporte de 2001 (ACM/IEEE-Curriculum, 2001) reafirma la conveniencia de aplicar el modelo de programar-primero, esto es, enseñar a programar en cursos introductorios, y provee una guía para implementarlo describiendo tres estrategias: imperativo-primero, funcional-primero y objetos-primero. La estrategia objetos-primero implica iniciar el curso con las nociones de objetos y herencia, probándolas mediante programas interactivos simples, e introducir después las tradicionales estructuras algorítmicas de control, pero siempre en el contexto del diseño orientado a objetos. El reporte reconoce la importancia que ha adquirido esta estrategia tanto en la academia como en la industria (ACM/IEEE-Curriculum, 2001, capítulo 7, página 30). El reporte más recientemente emitido en 2005 (ACM/IEEE-Curriculum, 2005) aborda y engloba la diversidad de programas relativos a la computación: Ciencias de la Computación, Ingeniería en Computación, Ingeniería de Software, Sistemas de Información, y Tecnología de Información, caracterizando la identidad que comparten a través de elementos concretos como el entendimiento de la programación que incluye los algoritmos y las estructuras de datos, el entendimiento básico del hardware y de los principios del diseño del software, así como el entendimiento de lo que pueden o no hacer los sistemas computacionales utilizando la tecnología actual.

Después del reporte emitido por la ACM en 2001, prevaleció la controversia acerca de dónde y cómo introducir las clases y los objetos en el primer curso de computación. El grupo de interés especial en la educación en Ciencias de la Computación de la ACM (SIGCSE) ofrece un foro para que los educadores discutan aspectos relacionados con el desarrollo, la implementación, y/o la evaluación de los programas de computación y del contenido de sus cursos. En marzo de 2004, gran parte de la discusión en este foro fue relativa al orden y grado de énfasis de los temas en el curso introductorio utilizando un lenguaje orientado a objetos como Java (Bruce, 2004). Al enfoque tradicional le llamaron objetos-después, y consiste en impartir los primeros dos tercios del curso de la misma manera en que se enseña un lenguaje procedural como C, introduciendo las bases de la orientación a objetos sólo durante las últimas semanas del curso. En cambio, el enfoque alternativo, objetos-primero, enfatiza desde el inicio del curso los conceptos de clase, objeto, encapsulamiento, comportamiento de los objetos, envío de mensajes, e invocación de métodos. Los proponentes del enfoque objetos-después argumentaron que las estructuras algorítmicas selectivas y repetitivas son aspectos críticos a los que se les restaría importancia y dedicación de seguir la aproximación de objetos-primero, a lo que los postulantes de este enfoque respondieron que tales conceptos serían abordados en otro momento y de otra manera al estudiar los métodos de los objetos.

El acuerdo universal al que se llegó en aquel foro de SIGCSE (Bruce, 2004) fue el reto que implica tanto la enseñanza temprana de los objetos como el aprendizaje de los mismos (Cooper, Dann y Pausch, 2003; Ventura, 2005; Forte y Guzdial, 2005; Woszczynski, Guthrie y Shade, 2005; Mannila, Peltomäki y Salakoski, 2006; Eckerdal, McCartney, Moström, Ratcliffe, Sanders y Zander, 2006). Si aprender a programar es una tarea difícil para los estudiantes de carreras de computación, lo es aún más para los alumnos de otras carreras, pues los primeros tienen una ventaja motivacional al haber elegido aprender a programar. La falta de motivación en los alumnos contribuye a la ocurrencia de problemas típicos como las calificaciones bajas o reprobatorias al final del curso, y la deserción del curso o incluso el cambio de carrera.

Experiencias de enseñanza y aprendizaje

Diversos estudios han sido desarrollados para explicar la dificultad en la enseñanza y el aprendizaje de los objetos de manera temprana identificando aspectos que pudieran predecir el éxito del curso introductorio de computación. En particular, Ventura (2005) descartó el género como un posible factor de desempeño de los estudiantes, y encontró que la experiencia

previa en programación y la cantidad de cursos de matemáticas que los alumnos hayan tomado previamente pueden predecir el éxito en cursos tradicionales de computación que siguen el enfoque objetos-después, pero no en cursos que abordan los objetos-primero.

Ragonis y Ben-Ari (2005) analizaron por dos años el desempeño de estudiantes de preparatoria al aprender aspectos del paradigma orientado a objetos como modularidad, encapsulamiento y ocultamiento de información utilizando Java y BlueJ. Específicamente, experimentaron introduciendo en el siguiente orden los conceptos de clase y objeto utilizando diagramas en UML, la invocación de métodos en BlueJ, la programación de clases en Java incluyendo constructores, métodos de acceso y otros métodos que implicaran expresiones y asignaciones simples, y la construcción de clases compuestas de objetos como atributos. Una vez que los alumnos lograban entender tales temas, se abordaban las estructuras de control y los arreglos. Durante el curso, los autores encontraron concepciones erróneas por parte de los estudiantes en los siguientes rubros: el orden en que se ejecutan las acciones, la relación entre una clase y los objetos de la clase, la relación entre la creación del objeto y la ejecución del constructor de la clase, y la diferencia entre clases con atributos de tipos simples y clases que incluyen además objetos como atributos. No obstante, concluyeron que el orden en que los conceptos fueron presentados y la simplicidad del ambiente de desarrollo BlueJ permitieron a los estudiantes asimilar las bases de la orientación a objetos, lo cual demostraron durante la planeación e implementación de sus proyectos finales.

La literatura da cuenta de experiencias diversas sobre la impartición del primer curso de programación en preparatorias, institutos tecnológicos y universidades. Mannila y colaboradores (2006), por ejemplo, proponen el uso de un lenguaje simple que actúe como facilitador del aprendizaje, considerando que los estudiantes no sólo tienen el reto de aprender a resolver problemas computacionales, sino también de aprender a expresar sus soluciones observando la sintaxis y semántica de un lenguaje de programación, y aunque éste no sea el aspecto fundamental del curso, sí juega un rol importante. El estudio presentado por los autores consistió en analizar programas escritos en Java y programas escritos en Python por estudiantes de preparatoria, con el fin de determinar si los errores que cometieron o las dificultades que encontraron eran atribuibles al lenguaje empleado. En una segunda parte del estudio, se dio seguimiento durante el primer semestre en la universidad a los estudiantes que programaron en Python en la preparatoria y que ahora programaban en Java, con el propósito de explorar la transición de un lenguaje simple a uno avanzado. Los resultados indicaron un mayor número de errores de sintaxis en los programas escritos en Java, y una mayor proporción de programas escritos en Python que ejecutaban correctamente y cumplían con las especificaciones. En las entrevistas realizadas durante la segunda parte del estudio, los alumnos distinguieron entre aprender a programar y aprender un lenguaje de programación, afirmando que "la idea de programar es la misma sin importar el lenguaje que se use" (Mannila et al., 2006), y que el haber utilizado Python previamente les había dado una enorme ventaja durante su primer curso con Java en la universidad.

Herramientas y métodos de soporte al aprendizaje

Experiencias reportadas en la literatura involucran el uso de ambientes de desarrollo (IDEs) pedagógicos para Java como BlueJ (Ragonis y Ben-Ari, 2005), y micromundos como Alice (Cooper et al., 2003). BlueJ (Kölling, Quig, Patterson y Rosenberg, 2003) provee un ambiente en donde el usuario inicia con un conjunto predefinido de clases, puede crear objetos e invocar sus métodos para ilustrar su comportamiento, y la estructura del proyecto es presentada gráficamente al estilo UML. Alice (Cooper et al., 2003) por su parte ofrece un ambiente interactivo de programación y animación en 3D que permite a los estudiantes construir mundos virtuales, usar/modificar objetos 3D, y escribir programas para generar animaciones. El sistema Alice fue diseñado y desarrollado en la Universidad Carnegie Mellon, donde ha sido utilizado con éxito desde el año 2000 en el primer curso de programación antecediendo la enseñanza de Java. Cooper et al. (2003) reportan haber desarrollado en los estudiantes fortalezas como asimilar la importancia del diseño previo a la implementación del programa, y las nociones de objeto y clase, así como obtener una buena intuición acerca del encapsulamiento, el concepto de método, y el sentido de la herencia. Concretamente, los autores presentan resultados derivados de utilizar Alice en un curso introductorio previo al primer curso de computación (CS1) impartido en Ithaca College y Saint Joseph University durante el año escolar 2001-2002. De 21 alumnos, 11 tomaron el curso con Alice y 10 no. El grupo de prueba de 11 estudiantes obtuvo calificaciones en CS1 significativamente mejores que las reportadas por el grupo de control de 10 alumnos. El 91% del grupo de prueba continuó con el segundo curso de computación (CS2), mientras que sólo el 10% del grupo de control lo hizo.

Nuutila y colaboradores (2005) relatan su experiencia desde el año 2000 al aplicar el método de aprendizaje basado en problemas (PBL) en un curso introductorio de programación para alumnos del primer semestre de la carrera Redes de Información en la Universidad Tecnológica de Helsinki. El objetivo de este curso es aprender la orientación a objetos con Java siguiendo el enfoque de objetos-primero con el fin último de que los estudiantes puedan implementar de manera independiente aplicaciones no triviales utilizando interfaces gráficas de usuario. El método PBL se compone de 7 pasos que

pretenden reforzar el aprendizaje a través de recurrir a conocimiento previo de los alumnos que se relacione con los nuevos temas a estudiar, asociar estos temas con situaciones específicas que pueden ocurrir en la realidad, y hacer que los estudiantes implementen el material aprendido. Los autores implementaron PBL por medio de reuniones semanales del grupo de 30 estudiantes para discutir casos reales. Cada caso presenta un tema que requiere una explicación o solución y que al final conduce al grupo a definir metas de aprendizaje referentes a aspectos importantes de la programación. En el semestre los alumnos procesan 10 casos, además de resolver prácticas de programación y un proyecto final. Los resultados de encuestas aplicadas semestralmente a los alumnos indicaron que las sesiones PBL son un mecanismo efectivo que mejora la motivación para aprender conceptos abstractos, fomenta el trabajo en grupo, propicia el uso de métodos creativos como la lluvia de ideas, enfatiza el proceso de planeación de un trabajo, entrena a los alumnos en sus habilidades de comunicación y argumentación, libera la tensión que pueda propiciar el curso, y promueve la búsqueda de información de distintas fuentes. Además, como beneficio medible, los autores reportan un descenso en la tasa de deserción de los alumnos del 45% al 17%. La universidad ha ofrecido también el curso introductorio de programación sin aplicar el método PBL. Tanto los egresados de este curso tradicional como los del curso con PBL toman en el segundo semestre un curso avanzado de programación con Java. De acuerdo con los autores, las calificaciones que en promedio han obtenido los alumnos provenientes del curso con PBL son ligeramente superiores a las del resto de los estudiantes.

DISEÑO DEL NUEVO CURSO

La primera decisión que se tomó al rediseñar el curso fue iniciar con el paradigma de programación orientada a objetos y después introducir las estructuras algorítmicas de control de flujo, es decir, se eligió la estrategia primero-objetos. Se pretende entonces lograr que los alumnos de Ingeniería sean capaces, al finalizar el curso, de desarrollar programas interactivos de mediana complejidad construidos mediante la cooperación de múltiples objetos. Para contribuir al logro de este objetivo se decidió incluir en la impartición y evaluación del curso la realización de tres proyectos (interesantes) de desarrollo de software, que promueven también el trabajo en equipo.

Para introducir las ideas de análisis, diseño e implementación de programas utilizando objetos que interactúan entre ellos se decidió emplear Alice (Dann, Cooper y Pausch, 2009) y posteriormente concretar estas ideas en programas escritos en Java utilizando un ambiente de desarrollo integrado comercial.

Una gran ventaja de introducir la programación de esta forma es que Alice es un software libre educativo (www.alice.org) que proporciona un ambiente de programación en el que diferentes objetos interactúan en un mundo virtual 3D interpretando una historia y el programa se ve funcionando (y/o fallando) desde la primera sesión del curso. Alice se utiliza en la primera parte del semestre para proporcionar una introducción muy sencilla y visual de los principales conceptos de la programación orientada a objetos: el ambiente de programación cuenta con una galería de clases a partir de la cual se instancian objetos mediante un solo *click* del ratón, y los objetos encapsulan atributos y métodos comunes a personajes de las historias. Los alumnos modularizan naturalmente sus programas (en escenas y tomas), y practican de esta manera un enfoque de programación incremental basado en métodos.

Los estudiantes agregan atributos y programan nuevos métodos en los objetos de Alice para especializar su funcionamiento. Esta codificación es muy sencilla ya que no involucra teclear instrucciones siguiendo reglas gramaticales estrictas, sino que se basa en un formateo visual muy simple (Cooper et al., 2003). Los objetos especializados pueden guardarse como una nueva clase que los alumnos importan posteriormente en otros mundos. La creación de nuevas clases a partir de las existentes introduce de manera natural el concepto de herencia, aunque los temas formales de herencia y de polimofismo, así como el de recursividad, se estudian en el curso de *Estructuras de Datos para Ingeniería*.

Alice se utilizó en el curso también para introducir el concepto de arreglos y sobre todo el uso de listas para almacenar objetos. El segundo proyecto de Alice en el curso requería el uso de listas y de programación dirigida por eventos, para ofrecer interactividad con el usuario en las historias.

Aún cuando el código de los métodos puede ser desplegado en Alice al estilo Java, pasar del desarrollo de animaciones en este ambiente a la programación de aplicaciones en Java presenta el reto, principalmente, de la menor amigabilidad del lenguaje y del ambiente de programación. Para simplificar esta transición es necesario ligar ambas partes reutilizando todos los conceptos introducidos previamente en Alice al enseñar a programar en Java. Por ejemplo, la estructura de galerías y mundos de Alice se traduce, durante la segunda parte del curso, en un paquete de clases simples en Java (galería de clases) que los alumnos programan y posteriormente utilizan en un paquete de pruebas y aplicaciones (mundos ejecutables). Es importante mencionar que la funcionalidad de los métodos de cada clase en la galería se prueba de manera unitaria, utilizando *asserts*, antes de emplear a los objetos en aplicaciones.

Conforme transcurre el semestre, las aplicaciones desarrolladas por los alumnos son cada vez más complejas e incluyen la manipulación de arreglos genéricos (incluyendo diversos algoritmos de ordenamiento y búsqueda) y la construcción de una lista, con la funcionalidad vista en Alice, usando estos arreglos. Para hacerlas más reales y simplificar el proceso de pruebas, las aplicaciones hacen uso de datos (en algunos casos reales) almacenados en archivos de texto. La interacción entre el usuario y la aplicación Java se realiza primero mediante el teclado y la consola, y posteriormente se desarrollan interfaces gráficas, utilizando ventanas, etiquetas, campos y áreas de texto, botones y listas. Los conceptos de subclases (*extends*) e interfaces (*implements*) de Java sólo se introducen al programar las interfaces gráficas de usuario.

Como resultado del rediseño llevado a cabo, al final del curso los alumnos son capaces de realizar proyectos más grandes e interesantes que los que desarrollaban en semestres anteriores.

Un componente importante de nuestro primer curso de programación siempre ha sido la parte de laboratorio en la que los alumnos practican los conceptos vistos en teoría bajo la supervisión y guía del profesor. El curso se desarrolla, la mitad de las horas semanales, en un aula con computadoras personales, y cuenta con un cuaderno de trabajos prácticos que deben desarrollarse utilizando las herramientas apropiadas. Siguiendo un enfoque ingenieril, durante el curso los alumnos deben seguir normas de construcción de diagramas de flujo y de estilo de programación, las cuales se encuentran disponibles para los alumnos en un sistema Web propietario denominado *comunidad itam*.

Como entorno de programación para Java seleccionamos Eclipse, ya que es un ambiente de desarrollo (libre) que proporciona muchas facilidades de codificación y que se utiliza en proyectos complejos. Con esta elección los alumnos utilizan desde el primer semestre una herramienta sólida, aunque más compleja que otras (como JCreator). Los alumnos bajan e instalan en sus propias computadoras todo el software necesario y aprenden el manejo de los ambientes de Alice y Eclipse. En las sesiones de laboratorio los profesores muestran a los alumnos sólo algunas características de los entornos de programación, y es la curiosidad de los estudiantes la que los lleva a descubrir por sí mismos aspectos como la generación automática de código: constructores, *getters* y *setters*, *main* y *extends*, que aceleran y facilitan la escritura de programas al disminuir los errores de codificación.

RESULTADOS

El nuevo curso se impartió en 6 grupos, con 27 alumnos en promedio por grupo, durante el semestre agosto – diciembre 2008. Como parte del proceso de mejora continua de la División Académica de Ingeniería, al término del semestre evaluamos qué tan bien se cumplieron los objetivos del curso y cuál es la contribución de éste al cumplimiento de los *outcomes* ABET esperados en los egresados de programas de Ingeniería (www.abet.org).

Basándonos en la experiencia de Craig y Aburdene (2005) se elaboró la tabla de la figura 1 para evaluar cómo la nueva materia de *Algoritmos y Programas* influye para alcanzar esos *outcomes*. En los renglones de la tabla se listaron los objetivos de aprendizaje de la materia. En las columnas, identificadas con las letras "a" a la "k", aparecen las características ABET deseadas en el egresado. Con una X en la posición i,j se indica que el objetivo del renglón i contribuye a alcanzar la meta de la columna j. Como se puede observar en la tabla, la mayoría de los objetivos planteados del curso se relacionan con el análisis del problema, la identificación de datos y resultados, la identificación y diseño de una solución algorítmica del problema, la abstracción y la búsqueda de eficiencia. Los 16 puntos que aparecen en la tabla son los más importantes, sin embargo no son los únicos.

Los instrumentos que se utilizaron para medir el nivel del logro de los objetivos del curso son: una encuesta anónima a los alumnos y otra similar a los profesores, un examen final y los tres proyectos desarrollados durante el curso. En la tabla de la figura 1 se presentan los resultados de las mediciones que se hicieron con los instrumentos señalados. Los espacios vacíos de un renglón i, columna j deben interpretarse como la ausencia de medición del objetivo ubicado en el renglón i, por medio del instrumento de la columna j.

Para el examen final (columna Ex. Final) se tomó en cuenta el número de aciertos de cada una de las preguntas, considerando sólo a los alumnos que aprobaron la materia. A diferencia de la encuesta, en el examen se pudo excluir los resultados de aquellos que reprobaron la materia y, por lo tanto, considerar sólo a los que la aprobaron ya que son los que tienen que ir reuniendo las características deseables al egresar. Para cada objetivo, se distribuyó un peso ($\sum pi = 1$) entre todas las preguntas del examen que tuvieran relación directa con dicho objetivo. Dicho peso se multiplicó por el coeficiente de aciertos (cAi) que se calculó como el total de respuestas correctas de la pregunta i entre el total de respuestas obtenidas para esa pregunta. Finalmente se sumaron todos los productos parciales obteniendo así una única puntuación ($\sum$ cAi * pi). Por ejemplo, el objetivo 2 fue medido por 4 preguntas del examen final. El cA de cada una de las preguntas fue 0.94, 0.79, 0,54 y 0,78; y los pesos fueron 0.15, 0.15, 0.3 y 0.4 respectivamente. Por lo tanto, la puntuación final obtenida por el objetivo a través del examen final fue 0.74. Finalmente, este valor se multiplica por 5 para llevarlo a la escala utilizada en las encuestas. Para

evaluar el desempeño de los alumnos en los tres proyectos desarrollados en el semestre (columna Proy.) sólo se tomaron en cuenta las calificaciones de los alumnos que aprobaron la materia.

Las encuestas, aplicadas al concluir el examen final permitieron cuantificar la percepción de los alumnos y de los profesores con respecto a qué tanto se habían alcanzado los objetivos del curso. Analizando las columnas correspondientes (Perc. Alums y Perc. Profs), se puede observar que las mayores diferencias se encuentran en los objetivos 6, 8, 9, 12 y 15 y, por la importancia de los mismos, exige una revisión por parte de los profesores. La tercera columna (Prom. por Objetivo), que resulta del promedio de las mediciones del examen final y de los proyectos, muestra claramente los objetivos (aquellos cuyos promedios sean inferiores a 4) sobre los cuales se requiere mejorar el desempeño del curso.

Haciendo un análisis comparativo de los promedios por objetivo (de procedencia objetiva) y los valores obtenidos por las encuestas (de procedencia subjetiva), se puede mencionar que los tres comparten un valor mayor o igual a 3. Sin embargo, hay objetivos en los cuales la percepción de los maestros (y de los alumnos) es mucho más alta que el valor obtenido por examen y proyectos, como en el caso del objetivo 13. En cambio, en otros objetivos se registró una situación distinta; es decir, la percepción de los profesores (y de los alumnos) fue mucho más baja que el promedio de examen y proyecto. Ejemplos son los objetivos 8 y 9. Esto está indicando que la idea que se tiene acerca del conocimiento impartido (y adquirido) no corresponde con la evaluación objetiva del mismo.

CONCLUSIONES

Al terminar el nuevo curso introductorio de Computación, los alumnos de Ingeniería adquirieron un buen dominio del paradigma de programación orientado a objetos, modelaron aplicaciones complejas que involucraban varias clases y las programaron con interfaces gráficas amigables. Los alumnos adquirieron nuevas habilidades analíticas que serán aprovechadas en un segundo curso.

Analizando los resultados generales obtenidos en la primera impartición del curso, en una junta de seguimiento con los profesores se obtuvieron las siguientes observaciones: 1) el paso de Alice a Java no resultó tan natural como se esperaba, en parte porque el uso de instrucciones selectivas de mediana complejidad y de arreglos en las historias de Alice no es muy intuitivo; y 2) faltó tiempo para desarrollar y reforzar más el pensamiento algorítmico de los alumnos, para analizar las soluciones a los problemas en búsqueda de lograr una mayor eficiencia y simplicidad, y para probar las soluciones a los problemas en búsqueda de lograr soluciones más robustas y completas.

En subsecuentes imparticiones del curso se dedicará menos tiempo al estudio de detalles particulares a Alice, para poder plantear una gama más variada de problemas a los alumnos y aumentar su capacidad de abstracción y de análisis. Esperamos que la siguiente versión de Alice, con menos *bugs* y que tiene personajes humanos con métodos más naturales para caminar y que permite grabar las animaciones desarrolladas para subirlas a You Tube, capte más el interés de los alumnos. En el desarrollo en equipo de programas en Alice se solicitará que los caracteres individuales especializados que necesita la historia sean construidos individualmente y luego importados en un mundo virtual para reforzar el concepto de herencia y reusabilidad. Asimismo, para aumentar la sensibilidad hacia el desarrollo de software de calidad se hará énfasis en el impacto de buenas y malas soluciones ingenieriles en las organizaciones y en la vida diaria.

Desde el punto de vista de retención de estudiantes, una encuesta anónima que contestan los estudiantes al final de cada materia mostró cualitativamente que el nuevo curso incrementa la satisfacción de los alumnos, pero que no cambia los requisitos de constancia, asistencia a clase y elaboración constante de tareas, necesarios para aprobar el curso. En el periodo en el cual se implementaron las modificaciones no se produjo ningún cambio favorable en el porcentaje de alumnos que no aprobaron el curso ni en los promedios de los alumnos que si lo hicieron, con respecto a los resultados históricos. Dado que después de dos semestres impartiendo el nuevo curso se tiene ahora una planta de profesores con experiencia, un temario detallado clase por clase y un cuaderno de ejercicios depurado, al término del semestre agosto – diciembre de 2009 estaremos en condiciones de comparar estadísticamente los resultados obtenidos, para determinar si existen diferencias significativas en el aprovechamiento de los alumnos con respecto al curso anterior.

Objetivos de aprendizaje de la materia	*Outcomes* ABET											Ex. Final	Proy.	Promedio	Perc. Alums	Perc. Profs
	a	b	c	d	e	f	g	H	i	j	k					
1. Identificar los datos de entrada y de salida de un problema, a partir de la descripción del mismo.		X	X		X		X					3,77	4,3	**4,03**	4,54	*4,83*
2. Identificar las clases que intervienen en un problema, a partir de la descripción del mismo.		X	X		X			X				3,68	4,3	**3,99**	4,49	4,50
3. Identificar del conjunto de datos los miembros de una clase.			X									3,40	4,3	**3,85**	4,61	4,50
4.Representar una clase por medio de un lenguaje gráfico formal.											X		4,1	**4.1**	4,31	4,50
5. Identificar la secuencia de actividades o pasos necesarios para resolver un problema.		X	X		X							3,33	4,2	**3,77**	4,11	4,00
6. Formular soluciones algorítmicas a problemas.	X		X		X							3,14	4,2	**3,67**	4,18	3,17
7. Utilizar las estructuras algorítmicas de control correctamente.												3,21	4,8	**4,01**	4,09	3,83
8. Probar la solución algorítmica para asegurarse que cubre todas las especificaciones, por medio de mapas de memoria.		X			X			X				3,69	4,3	**4,00**	3,54	3,00
9. Buscar alternativas para mejorar la eficiencia del algoritmo diseñado.					X			X					4,2	**4,20**	3,80	3,00
10. Programar la solución algorítmica empleando un lenguaje de programación orientado a objetos.			X		X						X	3,50	4,7	**4,10**	4,25	4,33
11. Probar el programa para determinar si el resultado generado es el esperado.		X	X		X			X				4,54	4	**4,27**	4,29	4,17
12. Buscar alternativas para mejorar la eficiencia del programa generado.					X			X					4	**4,00**	3,85	3,17
13. Determinar la estructura de datos más adecuada al problema.	X		X					X				2,72	4,4	**3,56**	4,15	4,33
14. Utilizar correctamente las estructuras de datos.	X											2,85	4,2	**3,53**	4,04	4,17
15. Documentar el programa adecuadamente.							X					3,62	4,4	**4,01**	4,14	3,67
16. Nombrar las variables y métodos con nombres adecuados.							X					3,62	4,7	**4,16**	4,49	4,50

Figura 1. Consolidado de mediciones

REFERENCIAS

1. ACM/IEEE Curriculum 2001 Joint Task Force (2001) Computing Curricula 2001 - Computer Science, IEEE Computer Society Press and ACM Press, December, http://www.acm.org/education/curricula.html
2. ACM/IEEE Curriculum 2005 Joint Task Force (2005) Computing Curricula 2005 - Computer Science, IEEE Computer Society Press and ACM Press, September, http://www.acm.org/education/curricula.html
3. Bruce, K. (2004) Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list, *Bulletin of the ACM Special Interest Group on Computer Science Education* (SIGCSE), 36, 4, 29-35.
4. Cooper, S., Dann, W., and Pausch, R. (2003) Teaching objects-first in introductory Computer Science, *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, February 19-23, Reno, Nevada, USA, 191-195.
5. Craig, S. and Aburdene, M. (2005) A Tool for Consolidating Results from Learning Assessment, *Proceedings of the 2005 American Society for Engineering Education (ASEE) Annual Conference & Exposition*, June 12-15, Portland, Oregon, USA.
6. Dann, W., Cooper, S., and Pausch, R. (2009) Learning to program with Alice. Upper Saddle River, NJ: Prentice-Hall.
7. Eckerdal, A., McCartney, R., Moström, J-E., Ratcliffe, M., Sanders, K., and Zander, C. (2006) Putting threshold concepts into context in Computer Science education, *Proceedings of the Eleventh Annual Conference on Innovation and Technology in Computer Science Education* (ITiCSE), June 26-28, Bologna, Italy, 103-107.
8. Forte, A., and Guzdial, M. (2005), Motivation and nonmajors in Computer Science: identifying discrete audiences for introductory courses, *IEEE Transactions on Education*, 48, 2, 248-253.
9. Hemmendinger, D. (2007) The ACM and IEEE-CS guidelines for undergraduate CS education, *Communications of the ACM*, 50, 5, 46-53.
10. Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. (2003) The BlueJ system and its pedagogy, *Computer Science Education*, 13, 4, 249-268.
11. Mannila, L., Peltomäki, M., and Salakoski, T. (2006) What about a simple language? Analyzing the difficulties in learning to program, *Computer Science Education*, 16, 3, 211-227.
12. Nuutila, E., Törmä, S., and Malmi, L. (2005) PBL and computer programming – the seven steps method with adaptations, *Computer Science Education*, 15, 2, 123-142.
13. Ragonis, N., and Ben-Ari, M. (2005) A long-term investigation of the comprehension of OOP concepts by novices, *Computer Science Education*, 15, 3, 203-221.
14. Ventura, P. (2005) Identifying predictors of success for an objects-first CS1, *Computer Science Education*, 15, 3, 223-243.
15. Wing, J. (2006) Computational Thinking, *Communications of the ACM*, 49, 3, 33-35.
16. Woszczynski, A., Guthrie, T., and Shade, S. (2005) Personality and programming, *Journal of Information Systems Education*, 16, 3, 293-299.