La enseñanza de la programación según un enfoque de Currículum Invertido

Licenciado José Albert Cruz Almaguer*
Licenciado Maykell Sánchez Romero**

Departamento de Técnicas de Programación

Universidad de las Ciencias Informáticas

*Facultad 1 - jalbert@uci.cu
**Facultad 7 - maykell@uci.cu

Introducción

La enseñanza de la programación constituye un enorme desafío a nivel mundial. "Aprender a programar es un objetivo clave en la gran mayoría de los cursos introductorios de carreras de computación; muchos educadores se preguntan si sus estudiantes están adquiriendo las habilidades necesarias en los mismos" [1] En nuestra Universidad es una triste realidad que luego de concluido un curso introductorio de programación la mayoría de los estudiantes no han vencido importantes objetivos de dichos cursos. Operar cambios en el diseño de los mismos se impone como una necesidad. El problema que tratamos es: ¿Cómo mejorar la enseñanza de la programación?

Tradicionalmente los cursos introductorios de programación desarrollaron los contenidos en una forma bottom-up: comienzan con los componentes más básicos de los lenguajes de programación tales como los tipos de datos básicos, las variables y la asignación; luego continúan con las estructuras de control, introducen la noción de subprogramas y finalmente (no en todos los casos) tratan principios y técnicas de estructuración y diseño de grandes sistemas. De esta forma muchos profesores tienden "...inconscientemente a aplicar una idea que en su día fuera popular en la biología: la ontogenia (la historia del individuo) repite la filogenia (la historia de la especie)" [2].

Durante los últimos tiempos, en la medida que se ha ido imponiendo el paradigma orientado a objetos y a crecido la necesidad de diseñar grandes sistemas, ha prosperado la idea de reordenar estos tópicos. Según el enfoque top-down se comenzaría con construcciones de muy alto nivel (tales como objetos y clases) para diferir las lecciones de construcciones a bajo nivel (asignación, estructuras de control).

Un nuevo enfoque ha aparecido: el Currículum Invertido, basado en la idea de una introducción de los tópicos similar al enfoque top-down pero siguiendo una estrategia outside-in (de afuera hacia adentro). En [9] se da una explicación detallada sobre el mismo. También se puede consultar [12] y [10].

Como objetivos del trabajo nos hemos propuesto:

- Analizar los métodos de enseñanza de la programación.
- Exponer la estrategia outside-in del enfoque de Curriculum Invertido como alternativa novedosa.

Este trabajo se inserta en el esfuerzo por aplicar el enfoque Currículum Invertido a la enseñanza de los cursos introductorios de programación en la UCI. Mediante la aplicación del mismo se espera mejorar el aprendizaje de las habilidades y conocimientos a adquirir y/o desarrollar en un curso introductorio de programación.

[1] Report by the ITiCSE 2001 Working Group on Assessment on Programmming Skills of First-year CS Students[11]

[2] Bertrand Meyer. Construcción de Software Orientado a Objetos[8]

Sobre la estructura del trabajo:

En la sección *La nueva estrategia de un enfoque* se desarrolla el trabajo. Primero se fundamenta debidamente el tema y del objetivo que se propone haciéndo un análisis de los métodos traicionales. Se hace un análisis crítico de la bibliografía utilizada y argumentación de la posición que asumen los autores. Se hace una descripción, análisis y valoración de la propuesta mostrando las bondades de la misma. A continuación se concluye haciendo referencia hacia la factibilidad de la implementación del enfoque en la UCI. Después se hacen las recomendaciones para darle continuidad al trabajo en aras de su puesta en práctica.

La nueva estrategia de un enfoque

Tradicionalmente en la enseñanza de la programación se han seguido dos estrategias para la introducción de los tópicos: la estrategia Bottom-up y la Top-down.

Inicialmente los cursos se implementaban de una forma Bottom-up (de abajo hacia arriba): se comenzaba con las construcciones de bajo nivel de los lenguajes de programación, como tipos básicos, conceptos como el de variable, y en la medida en que avanzaba el curso se iban añadiendo construcciones cada vez de más alto nivel, si el tiempo lo permitía. A continuación se ilustra la estrategia bottom-up a través del breve análisis de un libro.

Bottom-up

Caso de estudio: Structure and Interpretation of computer programs

Este libro [1] ha sido usado en el Massachusetts Institute of Technology (MIT) desde 1980 para el curso de introducción a la programación, está basado en el lenguaje Scheme y en sus tres primeros capítulos se concentran los conceptos básicos de la programación:

> Capítulo 1: *Construyendo abstracciones con procedimientos.* Introduce las construcciones más básicas que permitan lograr programas numéricos simples: comienza con expresiones, variables y condicionales, para luego introducir la recursión y la iteración.
>
> Capítulo 2:*Construyendo abstracciones con datos.* Se lleva la atención desde las rutinas hacia los datos, primero conceptualizando la abstracción de datos, para luego usar listas en la construcción de secuencias y estructuras jerárquicas para datos organizados de forma no lineal, a continuación analiza el procesamiento de información simbólica y finaliza con el tratamiento de operaciones genéricas.
>
> Capítulo 3:*Modularidad, objetos y estado.* Trata principios de diseño de grandes sistemas.

En este texto vemos un ejemplo claro de un orden de introducción de contenidos bottom-up: comienza con los elementos básicos como las expresiones y variables, luego trata los procedimientos para entonces introducir los objetos y la modularidad.

El mismo desarollo de la industria del software y el surgimiento del paradigma Orientado a Objetos puso a relieve las grandes limitaciones que tenía esa forma de enseñanza dada la necesidad de que los profesionales adquirieran técnicas de estructuración y diseño de grandes sistemas, por lo que se decidió invertir la introducción de los tópicos, dando lugar a la estrategia Top-down. Veamos, mediante el análisis del desarrollo de los contenidos de un libro como se desarrollaría un curso siguiendo una vía top-down.

Top-down

Caso de estudio: Big-Java

El libro Big-Java [5], presenta una división del contenido en cuatro partes:

I . Capítulos del 1 al 8, cubre lo básico de la orientación a objetos, comenzando por la definición de hardware y software, y el típico hello word, en este primer capítulo se cubre el llamado a métodos. En el segundo capítulo se introduce superficialmente las nociones de objetos y clases, junto a la creación de objetos, variables, declaración e implementación de clases, e interfaces; al fin de este tópico se debe haber ganado la habilidad de construir clases simples. El siguiente capítulo trata los tipos de datos primitivos e introduce la asignación, las constantes y los métodos estáticos. El capítulo 4 se encarga de los applets, los siguientes dos capítulos se enseña las estructuras de control. El séptimo profundiza en el diseño de clases y finaliza la primera parte enseñando técnicas de verificación de programas.

II . En la segunda parte (el capítulo 9) se discute las interfaces y el polimorfismo, seguido de la programación guiada por eventos y la herencia; el capítulo 12 analiza el desarrollo de interfaces gráficas de usuario (GUI), a continuación se introducen los arreglos, seguido del tratamiento de excepciones y los streams. Como conclusión del tema se exponen algunos criterios sobre el diseño de grandes sistemas.

III . La tercera parte se encarga de los algoritmos estándares y de las estructuras de datos. Comienza con la recursión como metodología de resolución de problemas, para dar pie a varios algoritmos tradicionales de ordenamiento y búsqueda. A continuación (capítulo 19) se exponen las listas enlazadas, las pilas y las colas; y el resto del tema profundiza en otras estructuras de datos.

IV . La cuarta y última parte cubre técnicas avanzadas como programación multi-hilos (multi-threading), de bases de datos y de red.

Dado el cúmulo de contenido solo serían temas de un curso introductorio el I y el II. El hecho de que antes de dar contenidos de bajo nivel, como la asignación y las condicionales, se traten objetos y clases simples muestra el diseño top-down.

Top-down *vs* bottom-up con orientación a objetos

Enseñando introducción a la programación de una forma bottom-up se centra en "programación a lo pequeño". Este modo de ver el problema es opuesto a la realidad de la práctica, en él los estudiantes se pierden importantes conceptos. El enfoque top-down se centra por su parte en "programación a lo grande", da a conocer importantes técnicas OO y de ingeniería de software en oposición a la enseñanza atada a detalles sintácticos dependientes de lenguajes a de programación. En este caso los estudiantes cuando acaban el curso tienen un

mejor entendimiento y apreciación de cosas como diseño top-down, abstracción de datos y de subprogramas, ocultación de información y componentes software reusables [4].

Decker y Hirshfield [3] consideran que el enfoque bottom-up no enfatiza los tan importantes conceptos de reusabilidad, ocultación de información y estructuración clara de soluciones. Los estudiantes no pueden ser motivados por cosas que no parecen ser importantes: la necesidad de subprogramas, parámetros, ocultación de información y efectiva interfaz de usuario.

Grandes bibliotecas de código básico vs pequeñas

Otra cuestión a analizar para el desarrollo de un curso introductorio de programación es el uso de bibliotecas de clases prediseñadas.

Dado el aumento de tamaño que han experimentado los programas se ha impuesto como una necesidad el uso marcos de trabajo (frameworks) que contengan componentes prefabricados que permitan lograr aplicaciones con poco esfuerzo, reforzando la visión de "programar en lo grande". Trabajar de esta manera en un curso introductorio de programación presenta ventajas y desventajas.

Sin bibliotecas

La mayoría de los cursos introductorios de programación no usan código preelaborado, lo cual presenta la ventaja de que los ejercicios no plantean mucho código a analizar (son muchos los especialistas en computación que consideran que es mucho más difícil leer código que escribirlo), por lo que parece ser que escribir su propio código le es más fácil a los estudiantes; sin embargo, con este modo de trabajo sólo se pueden usar ejercicios introductorios sencillos.

Con grandes marcos de trabajo

En este caso los ejercicios a resolver se desarrollan modificando y/o extendiendo la plataforma (framework) misma. Tiene como ventajas el hecho de que se pueden diseñar ejercicios muy interesantes, que incluyan multimedia e interfaces gráficas; muestra ejemplos de componentes de software bien diseñados que les permiten a los estudiantes intentar imitar y los conducen a crear buenos hábitos de desarrollo de grandes sistemas. Por otra parte presenta el inconveniente de que si los estudiantes no se abstraen adecuadamente pueden confundirse con tanto código, y además le presenta mucho más trabajo al profesor, que está obligado a dominar a la perfección las técnicas que profesa pues pondría como material de estudio su propio trabajo.

Outside-in: el Currículo Invertido

A raíz de los constantes análisis en el área de la enseñanza de la programación a surgido una nueva metodología para guiar el desarrollo de la misma: el Curriculum Invertido con estrategia Outside-In.

El término ha sido acuñado por Bertrand Meyer y está basado en la idea de una introducción de temas que vaya de los más generales a los particulares (outside-in) y de una apertura progresiva de çajas negras"[7]. En este enfoque

es medular el uso de un extenso framework de código ya probado, así como la inclusión de mecanismos avanzados de programación orientada a objetos y de ingeniería de software. Más que introducir tópicos curriculares en dirección bottom-up o top-down, en el modo çurrículo invertido.enseña a programar .outside-in"(de lo general a lo particular).

La forma tradicional con que se imparten los temas en un curso introductorio de la programación es inicialmente con los bloques más básicos como las variables y la asignación, seguido de las estructuras de control y datos. De esta manera los estudiantes adquieren habilidades prácticas en la construcción de programas pero se marginan importantes conceptos de la construcción de sistemas que son de mucha importancia para el futuro trabajo profesional. Un profesional de la informática debe poseer una serie de habilidades en el mantenimiento y desarrollo de grandes y complejos sistemas de software.

El presente enfoque se sustenta en la premisa de que la manera más efectiva de aprender a hacer software es usando software de calidad con interfaces correctamente diseñadas. Este software sería brindado al estudiante en una biblioteca de clases (.dll) la cual estaría accesible en código fuente, proveyendo de un repositorio de alta calidad con modelos a imitar. Debemos tener en cuenta que lo único que deberán realmente entender los estudiantes son las interfaces que les permiten usarlas, de esta manera los iniciados serían capaces de lograr aplicaciones interesantes aún cuando inicialmente no hagan otra cosa que un par de llamadas a rutinas, en la medida que avance el curso se le irán dando elementos que le permitan al final construir sus propias bibliotecas.

En esta estrategia el alumno comienza como un consumidor para irse convirtiendo poco a poco en un productor, proceso en el que no se olvida la enseñanza de estructuras y habilidades de más bajo nivel sino que se les invierte el orden, haciendo énfasis en el desarrollo de habilidades de diseño.

La principal característica que debe poseer un profesional de la informática es la capacidad de abstracción : separar lo esencial de lo superfluo, lo perdurable de lo momentáneo, la especificación de la implementación [9]. En la práctica profesional abstraerse es una necesidad en función de poder solucionar tareas que por su tamaño obligan a usar lo que otros han hecho. Este enfoque, al permitir llegar a sistemas completos de manera fácil, usando para ello solamente interfaces bien diseñadas, desarrolla en los estudiantes de manera natural esta capacidad.

El software de soporte

Para implementar un cambio en la enseñanza siguiendo esta nueva manera es imprescindible un software de soporte, una biblioteca de clases sobre la cual se desarrolle el curso.

La selección del área de aplicación de la biblioteca debe hacerse con mucho cuidado. Entre otros elementos debe tener en cuenta:

- Ser familiar para los estudiantes, de manera que no necesitemos invertir tiempo en entender el contexto, y pudiendo dirigir todos los esfuerzos en la características de la programación en la creación de soluciones.

- Proveer de una gran colección de algoritmos y estructuras de datos interesantes, así como aplicaciones de los conceptos fundamentales.
- Uso de multimedia y sofisticadas GUIs[3].

Los estudiantes comienzan el curso corriendo y explorando una aplicación, para luego extenderla añadiendo simples llamadas a objetos predefinidos, lo que permite obtener resultados impresionantes desde el inicio del curso. Seguidamente es introducida la creación de objetos usando clases teniendo en cuenta solamente su interfaz, sólo luego de todo esto se iría revelando gradualmente la implementación.

Valoración de la propuesta

La propuesta de enseñanza de la programación con un enfoque de Curriculum Invertido y estrategia Outside-In presenta interesantes aspectos y es revolucionaria en su concepto, apartandose de las vías tradicionales pero tomando lo mejor de éstas (sobre todo dela Top-down), por lo que es valorada de muy positiva y con un gran potencial metodológico para la mejora de la enseñanza.

La puesta en práctica de la misma beneficia a los estudiantes pues:

- Hace énfasis en la "programación en lo grande", con lo cual van adquiriendo habilidades necesaria para enfrentarse a los grandes problemas que involucra a la industria del software actualmente.
- Los estudiantes encuentran primero las construcciones de alto nivel, con lo cual van aprendiendo correctamente el paradigma Orientado a Objetos y van ganando en poder de abstracción, pues el uso de la biblioteca de clases oculta al estudiante de los detalles innecesarios.
- Promueve desde el comienzo el dominio de conceptos de la ingeniería de software como lo son la abstracción, el reuso de código, el mantenimiento, la documentación, etc.
- Promueve el dominio del idioma inglés, pues las interfaces de comunicación estarán documentadas en inglés.

La propuesta es tan general que puede ser aplicada a otras asignaturas como Gráficos por Computadoras, P1, P2, P3, P4, Ingeniería de Software. Aplicada al departamento de Técnicas de Programación se puede considerar un proyecto unificador a largo plazo donde las asignaturas de la disciplina Programación se vean más estrechamente relacionadas, tributando de una forma más coherente a la formación profesional del estudiante.

Un gran peso en la propuesta recae en la biblioteca de clases. La misma tiene principalmente un fin pedagógico pero una buena biblioteca presenta más ventajas:

[3]GUI: Interfaz Gráfica de Usuario

- Ofrece perspectivas de patrocinio por parte de la industria si el dominio de aplicación es de interés directo para las empresas.
- "Dado que una biblioteca ambiciosa es un proyecto grande, existe un espacio para que contribuyan mucha personas, desde estudiantes universitarios avanzados hasta candidatos a maestros y doctores(...)"[4]
- Genera desafíos científicos que pueden dar lugar a publicaciones. La construcción de componentes de software reutilizables es uno de los desafíos más interesantes y difíciles de la Ingeniería de Software lo cual da lugar a tesis de maestrías y doctorados(ver [2]). Además, toda biblioteca se basa en una correcta taxonomia de los conceptos del dominio de aplicación, lo cual sugiere también la posibilidad de cooperación interdisciplinaria con investigadores de distintos dominios de aplicación.
- Se obtiene buenos resultados a corto y mediano plazo lo cual puede traer, de ser exitosa, mucho prestigio a la institución.

En [8] y [7] se valoran otras ventajas que trae la aplicación de la nueva metodología.

[4]Bertrand Meyer: Construcción de Software Orientado a Objetos.[8]

Conclusiones

El enfoque de Currículo Invertido acompañado de la estrategia Outside-in puede mejorar el aprendizaje de la programación. Aplicado a asignaturas de programación o a disciplinas completas relacionadas con la computación puede aportar grandes beneficios pedagógicos y productivos concretamente en el dominio de la Universidad de las Ciencias Informáticas. El enfoque tiene un carácter general por lo que puede ser introducido en otras asignaturas o disciplinas.

Referencias

[1] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 1996.

[2] Karine Arnout. *From Patterns to Components*. PhD thesis, ETH Zurich, 2004.

[3] R. Decker and S. Hirshfield. Top-down teaching: object-oriented programming in cs 1. In *Proceedings of the twenty-fourth SIGCSE technical symposium on Computer science education*. ACM Press, 1993.

[4] T. B. Hilburn. A top-down approach to teaching an introductory computer science course. In *Proceedings of the twenty-fourth SIGCSE technical symposium on Computer science education*. ACM Press, 1993.

[5] C. Horstmann. *Big Java*. John Wiley and Sons, 2001.

[6] Marcel Kessler. Exercise design for introductory programming. Master's thesis, ETH Zurich, 2004.

[7] Bertrand Meyer. Towards an object-oriented curriculum. 1993.

[8] Bertrand Meyer. *Construcción de Software Orientado a Objetos*. PRENTICE HALL, 1999.

[9] Bertrand Meyer. The outside-in method for teaching introductory programming. 2003.

[10] Bertrand Meyer. *Touch of Class*. 2004. Borrador.

[11] Danny Díaz Mark Guzdial Dianne Hagan Cary Laxer Lynda Thomas Ian Utting Tadeusz Wilusz Yifat Ben-David Kolikant Michael McCraken, Vicki Almstrum. Report by the iticse 2001 working group on assessment on programmming skills of first-year cs students. Technical report, 2001.

[12] Michela Pedroni. Teaching introductory programming with the inverted curriculum approach. Master's thesis, ETH Zurich, 2003.