ENSEÑANZA DE LA PROGRAMACIÓN BASADA EN ESQUEMAS

José Manuel Burgos Ortiz[1], Javier Galve Francés[1], Julio García Martín[1], Miguel Sutil Martín[2]

[1]*Profesores de la Facultad de Informática de la Universidad Politécnica de Madrid*
e-mail: *{jmburgos,jgalve,juliog}@fi.upm.es*

[2]*Profesor de Formación Profesional en Informática en el IES "Ciudad Escolar"*
e-mail: *msutil@arrakis.es*

RESUMEN: La mayoría de los cursos y textos de programación no resaltan el conocimiento clave de la programación ni prestan atención a las habilidades y destrezas que el experto maneja para escribir programas. Este trabajo presenta una exploración de ese conocimiento para dotarlo de estructura. Se estudia la idea de esquema de programación como unidad válida para organizar el complejo conocimiento de un primer curso de programación poniendo especial interés en las relaciones entre problema, solución y programa.

1.- MOTIVACION.

La realidad actual de los programas que realizan nuestros estudiantes nos muestra un panorama decepcionante: código ilegible, poco o mal documentado y con una fuerte tendencia a representar ideas "ingeniosas" que esconden malas decisiones de diseño. En muchos casos, el origen del problema hay que buscarlo en los primeros momentos del periodo de formación del candidato a programador.

La mayoría de los cursos y textos de programación asumen como enfoque incuestionable el de dejarse llevar por la exposición de todas las construcciones sintácticas que constituyen el lenguaje e ir así mostrando ejemplos de programas que hacen uso de ellas. En este enfoque, que denominaremos *guiado por la sintaxis*, se enseña a programar al mismo tiempo que se enseña un lenguaje de programación, con lo que se mezclan los conceptos de la computabilidad con la sintaxis del propio lenguaje. Los problemas se muestran como los ejemplos necesarios para ilustrar las construcciones sintácticas y las soluciones se presentan como programas acabados que resuelven los problemas. De este modo, esta colección de problemas deslabazados y programas de última versión se convierten en el material del que disponen los estudiantes para resolver problemas nuevos.

El presente trabajo pretende buscar el conocimiento clave de un primer curso de programación y dotarlo de una estructura que lo haga coherente y manejable. Nos basaremos en los conocimientos obtenidos por investigadores del aprendizaje, el análisis de diversos trabajos relacionados con éste y la experiencia de los autores.

2.- UN MODELO DE APRENDIZAJE DE LA PROGRAMACIÓN BASADO EN ESQUEMAS.

Los sicólogos cognitivos han establecido lo que se conoce como *teoría de esquemas* para explicar los procesos de aprendizaje [AKB79]. Según esta teoría, el conocimiento se estructura y organiza en nuestra mente mediante unidades o "trozos" de conocimiento llamados *esquemas*. Un esquema es un conjunto de conocimientos extraídos de repetidas experiencias que han sido agrupados debido a sus relaciones y su efectividad para resolver, con un aire común, diversos problemas. El aprendizaje se puede ver entonces como la construcción de conocimiento mediante la creación de nuevos esquemas o mediante la combinación de esquemas preexistentes.

Lo que distingue al experto del novicio en cualquier área del saber es que el conocimiento en la mente del experto está formado por una estructura de esquemas que los jerarquiza y ordena, permitiendo una fácil recuperación, una utilización versátil y fructífera y la atracción de otros esquemas para engranarlos y así formar entramados más amplios y ricos[CS73].

El conocimiento de los programadores expertos también ha sido caracterizado mediante el modelo cognitivo anterior como consistente en secuencias de *acciones estereotipadas*, perfiladas como abstracciones intermedias de solución. Estas acciones estereotipadas han sido tratadas, modeladas y nombradas de diversas maneras por autores distintos: *esquemas de programación* [GRI81], *plantillas* [CL92a] [CLS93], *planes* [SOL85], *soluciones enlatadas* [SOL86] o *patrones* [PROOO] [ETW96]. Diversos estudios sugieren que la representación del conocimiento en la forma de esquemas de programación son de gran ayuda para los estudiantes a la hora de escribir programas [AR85] [LD85].

3.- ORGANIZACION DE LOS ESQUEMAS.

El objetivo central de este trabajo es el de encontrar un repertorio de esquemas que reúna el rico y complejo conocimiento que se transmite en un primer curso de programación, y permita ser transmitido de manera efectiva a programadores noveles. Las investigaciones hechas sobre el modo en que se organiza en la mente el conocimiento de programación sugieren que los programadores novicios tienen representaciones más superficiales, o sintácticas, de este conocimiento, mientras que los expertos manejan estructuras más profundas, semánticas, abstractas y conceptuales. Cuanto más abstractas y conceptuales sean las categorías que se manejen, más pequeño será su número, su nivel de organización será más alto y permitirá detectar más fácilmente las similitudes profundas entre los esquemas. Otra cualidad fundamental que deben tener los esquemas es que estén cohesionados, es decir, que estén interconectados de diversas maneras para estructurar el conocimiento, pues esto ayuda extrordinariamente a asimilar, memorizar, e integrar lo aprendido, así como acceder a dicho conocimiento en nuestra mente y utilizarlo de manera más versátil y fructífera. Por último, otra característica deseable de los esquemas es que permitan y faciliten la generación de soluciones de mayor dimensión mediante la combinación de los esquemas, así como la incorporación de nuevos esquemas.

Categoría		Especificación del Problema
Solución Directa		El problema es igual o semejante a otros previamente resueltos o bien es una combinación sencilla de ellos.
Análisis de Casos		En el problema hay que considerar casos distintos. El resultado toma valores distintos en función de los datos de entrada.
Colecciones	*Acumulación*	El problema consiste en acumular una expresión aplicada a cada elemento de una colección. Se exige un recorrido de la colección.
	Búsqueda	El problema consiste en hacer una búsqueda para saber si algún elemento de una colección cumple una propiedad o para saber cuál la cumple. Se exige un recorrido de la colección.
	Extremales	El problema consiste en determinar qué elemento de una colección maximiza o minimiza una expresión. Se exige un recorrido de la colección.
	Construcción	El problema consiste en construir una colección de salida como resultado de aplicar una expresion a cada elemento de una colección de entrada. Se exige un recorrido de la colección.
Lectura y Escritura		El problema consiste en leer o escribir datos en un medio externo.

a) Una Taxonomía de Problemas.

El primer paso para conseguir la estructuración de los esquemas consiste en categorizarlos desde la perspectiva más abstracta posible: la de los problemas [BGG00b].

Los problemas anteriores son los problemas que podríamos calificar como básicos. A ellos habría que añadir otros más avanzados, como los que siguen:

Categoría	Especificación del Problema
Juntar	El problema consiste en tomar elemento a elemento de dos colecciones y generar otra colección formada por la combinación de ellas.
Separar	El problema consiste en tomar una colección y generar dos a partir de ella.
Ordenar	El problema consiste en ordenar una colección de elementos.

b) Principios, Pautas y Plantillas.

Hemos establecido los siguientes ámbitos para poder identificar y tratar separadamente los esquemas que resumen el conocimiento de cada parcela relativo a cada una de las fases del proceso de desarrollo (análisis, diseño, implementación y pruebas):

- Principios: Las premisas, enunciadas de la manera más sencilla posible. Con ellas establecemos un modelo aproximado o marco de trabajo para esta fase del desarrollo. Responden a la pregunta ¿qué tengo que tener en la cabeza a la hora de aplicar los esquemas?.

- Pautas: Los métodos, las orientaciones metodológicas que encauzan la manera de usar los esquemas. Son las que dan respuesta a la pregunta ¿cómo y cuándo aplico los esquemas?

- Plantillas: Las herramientas, los esquemas operativos que resumen el conocimiento más concreto de cada fase. Responden a la pregunta ¿qué repertorio de esquemas tengo para aplicar?

4.- ESQUEMAS DE PROGRAMACIÓN.

Los esquemas se expanden a lo largo de las etapas de desarrollo para recoger conocimiento específico de cada etapa. Hemos considerado que los *patrones de programación* son un buen modelo para expresarlos [BGG00a]. En los últimos cinco años, este campo ha tenido una fuerte expansión, extendiéndose su uso no sólo a la enseñanza de la programación [AST98] [CL99] sino a la comunidad de programadores y de ingenieros de software [GHJV95]. Su principal aportación es un vocabulario de diseño común y el fomento de la reutilización de soluciones a problemas comunes. Antes de enunciar los esquemas de cada etapa, se pueden establecer los principios y pautas generales comunes a todos ellos [GEH81].

Principio General:

- *Refinamiento Progresivo*: El problema se debe partir en subproblemas para que, una vez resueltos, por combinación, den solución al problema original.
- *Modelado con tipos y descomposición en subtipos.*

Pautas Generales:

- Los subproblemas se puedan resolver. Cada subprograma debe hacer una sola cosa.
- Un subproblema sea resoluble con el mínimo impacto posible sobre los demás.
- La solución de cada subproblema debe suponer menos esfuerzo que el problema original.
- La representación de los datos se debe posponer tanto como sea posible. Una buena pauta no siempre fácil de seguir es la de elegir la representación de los datos que haga el programa sencillo [KP78]. Agrupar y organizar los subprogramas que manejan el mismo tipo de datos

a) Esquemas de Análisis.

a.1) Principios.

- La programación es una actividad orientada a objetivos. [GRI81].
- Es necesaria una comprensión completa del problema antes de resolverlo.
- Los problemas están estructurados en una taxonomía de problemas, como "familias" de problemas.

a.2) Pautas.

- En esta etapa se debe centrar la atención sólo en lo que tiene que ver con el qué y posponer el cómo.
- Especificar los datos de entrada y de salida.
- Especificar las premisas de las que se parte y los objetivos a conseguir. Refinar y precisar las precondiciones y las poscondiciones todo lo que se pueda.
- Buscar jerarquías de datos: Las jerarquías de los datos guían las jerarquías del problema y sus subproblemas.
- Partir el problema en subproblemas y aplicar recursivamente los esquemas a cada subproblema.
- Intentar categorizar el problema en la taxonomía. Reformular las las precondiciones y las poscondiciones si es necesario para poder categorizar el problema.

a.3) Plantillas.

- *Análisis de los datos*: Una plantilla para describir los datos.
- *Problema como contrato*: Una plantilla para establecer las premisas y los objetivos del problema.

b) Esquemas de Diseño.

b.1) Principios.

- *Principio de reciclaje* [CL92b]: Se debe intentar partir siempre de plantillas preestablecidas. Se trata de ver si existe un problema *isomorfo* (análogo o semejante) que ha sido previamente resuelto para, en tal caso, delegar en él la solución por *invocación* (utilización) o *clonación* (copiar su implementación). En otros casos basta con reutilizar las ideas utilizadas para resolver problemas semejantes
- *Diseño Guiado por los Datos*: La estructura de los datos guía, en muchos casos, la estructura del programa.
- *Refinamiento iterativo*, mejoramiento progresivo o crecimiento progresivo[PAT90]: Una buena estrategia para resolver el problema consiste en simplificarlo, resolverlo y luego ampliar la solución para resolver el problema original mediante vuelta atrás y revisión.
- *Factorización*: Reemplazar expresiones que se repiten muchas veces por llamadas a un subprograma común .
- *Generalización-Especialización*: Buscar un problema más general (generalización) y luego particularizarlo o bien considerar este problema como un caso particular de otro más general que ya está resuelto (especialización). La técnica de *inmersión* se basa en esta estrategia.

b.2) Pautas.

- *Aplicación, adaptación o invento*: Si el problema ha sido categorizado aplicar el esquema de solución. Si el problema no ha sido categorizado en la taxonomía, adaptar la funcionalidad del esquema de la categoría que más se aproxime a él o intentar inventar un nuevo esquema. Si el problema no ha sido suficientemente partido en el análisis, partirlo ahora en subproblemas.
- *Combinación*: Combinar los esquemas de alguna de las siguientes maneras:
- *Composición*: A la manera de la composición de funciones de matemáticas.
- *Secuenciación*: Uno a continuación de otro.
- *Anidamiento*: Uno anidado dentro de otro.
- *Mezcla*: Aplicando los dos esquemas a la vez.

b.3) Plantillas.

<table>
<tr><th colspan="3">BASICAS</th><th colspan="3">AVANZADAS</th></tr>
<tr><th colspan="2">Plantilla</th><th>Variaciones</th><th>Plantilla</th><th colspan="2">Variaciones</th></tr>
<tr><td colspan="2" rowspan="2">Entrada-Proceso-Salida</td><td>Elemento a elemento</td><td rowspan="2">Juntar</td><td colspan="2">• Iterativa</td></tr>
<tr><td>Por Lotes</td><td colspan="2">Recursiva</td></tr>
<tr><td colspan="2" rowspan="2">Lectura y Escritura</td><td>Presentación– Lectura</td><td rowspan="2">Separar</td><td colspan="2">• Iterativa</td></tr>
<tr><td>Lectura de tipo
y Escritura de tipo</td><td colspan="2">Recursiva</td></tr>
<tr><td colspan="2">Solución Directa</td><td></td><td>Búsqueda</td><td colspan="2">Con saltos</td></tr>
<tr><td colspan="2">Análisis de Casos</td><td></td><td></td><td colspan="2">Binaria</td></tr>
<tr><td rowspan="8">Colecciones</td><td rowspan="2">Acumulación</td><td>• Iterativa</td><td rowspan="8">Ordenación</td><td rowspan="2">Inserción Directa</td><td>• Iterativa</td></tr>
<tr><td>Recursiva</td><td>Recursiva</td></tr>
<tr><td rowspan="2">Búsqueda</td><td>• Iterativa</td><td rowspan="2">Selección</td><td>• Iterativa</td></tr>
<tr><td>• Recursiva</td><td>Recursiva</td></tr>
<tr><td rowspan="2">Extremales</td><td>• Iterativa</td><td rowspan="2">Rápida</td><td>• Iterativa</td></tr>
<tr><td>Recursiva</td><td>Recursiva</td></tr>
<tr><td rowspan="2">Construcción</td><td>• Iterativa</td><td rowspan="2">Mezcla</td><td>• Iterativa</td></tr>
<tr><td>Recursiva</td><td>Recursiva</td></tr>
</table>

c) Esquemas de Implementación.

c.1) Principios

- El código del programa debe atenerse a la sintaxis del lenguaje de programación.
- La implementación es la expresión de la solución en el lenguaje de programación.
- *Principio de la Programación Culta* ("literate programming") [KNU84] [CL92b] [SOL86]: No basta con escribir código sintácticamente correcto. Es importante que se escriba pensando en lectores humanos, es decir, que sea fácil de leer, entender y modificar. Es importante que el código sea autocontenido, es decir, que por sí solo documente y explique al lector humano por qué el programa resuelve el problema dado.

c.2) Pautas.

- Traducir la solución al lenguaje de programación.
- Estructurar el código en piezas pequeñas.
- Permitir que el programa se pueda leer de arriba abajo.
- Usar identificadores significativos.
- Reglas de encolumnado y tipografía.

c.3) Plantillas.

Plantilla	Variaciones	Plantilla	Variaciones
Utilización	• **Constantes**	*Selección*	• **Alternativas**
	• **Tipos**		• **Rango de posibilidades**
	• **Variables**	*Iteración*	*Con contador*
	• **Funciones**		*Condicionada*
	• **Procedimientos**	*Recorrido*	• **Intervalos**
	Arrays		• **Lineal Simple(Arrays y strings)**
	Ficheros		• **Archivos**
	Punteros		• **Listas Enlazadas**

d) Esquemas de Pruebas.

d.1) Principios.

- Los problemas que han sido categorizados en la etapa de análisis conducen a un programa correcto.
- Los programas que no han sido categorizados deben ser validados mediante pruebas.

d.2) Pautas.

- Probar los posibles puntos débiles del programa usando casos típicos y extremos. Elaborar un rango de casos exhaustivo.
- Probar cada pieza del programa (subprograma) por separado, yendo de más pequeño a más grande, concentrándose en cada pieza por separado.
- Si hay estructuras de datos, probar que la construcción de las mismas sea correcta, pero separada del resto del programa.

5.- CONCLUSIONES.

En este trabajo hemos presentado una propuesta para dotar de estructura el conocimiento de un primer curso de programación y así mejorar la efectividad de la enseñanza y del aprendizaje en dicho curso. La propuesta se basa, por un lado en lo que saben los expertos y por otro en lo que se sabe acerca de cómo las personas aprenden, organizando el conocimiento en esquemas de programación divididos en principios, pautas y plantillas, específicos para cada una de las fases del desarrollo de un programa. Con estos esquemas se facilita el diseño de cursos, materiales curriculares y la creación de materiales de enseñanza complementarios. Esta propuesta no está cerrada ni acabada. Se pueden identificar otros esquemas y además, aunque muchos de los esquemas están ya expresados en forma de patrones, quedan otros por escribir.

REFERENCIAS.

[AKB79] J. R. Anderson, P. J. Kline y C. M. Beasley. A General Learning Theory and Its Application to Schema Abstraction. En G. H. Bower (ed.). *The Psychology of Learning and Motivation*, vol. 13, New York Academic Pres (1979), pp. 277-318.

[AR85] J. Anderson y B. Reiser. The Lisp Tutor. *Byte*, 10, 1985, pp. 159-175.

[AST98] O. Astrachan. Design Patterns: An Essential Component of CS Curricula, *SIGCSE Bulletin and Proceedings*, 30(1), March 1998, pp. 153-160.

[BGG00a] J.M. Burgos, J.Galve y J. García. From Problems to Programs: A Pattern Language to Go from Problem Requirements to Solution Schemas in Elementary Programming. *EuroPLoP'2000.*

[BGG00b] J.M. Burgos, J.Galve y J. García. Una Taxonomía de Problemas para la Enseñanza de la Programación, *Actas del VII Congreso Internacional de Informática en la Educación INFOREDU'2000*, Mayo 2000, La Habana (Cuba).

[CL92a] M. J. Clancy, M. C. Linn. Designing Pascal Solutions: A Case Study Approach. Computer Science Press, 1992.

[CL92b] M.J. Clancy y M.C. Linn. The Case for Case Studies of Programming Problems, *Communications of the ACM*, March 1992, 3, 3, pp. 121-132.

[CL99] M.J Clancy y M.C. Linn. Patterns and Pedagogy. Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education, 1999, 37-42.

[CLS93] M.J Clancy, M.C. Linn y P.K. Schank. Supporting Pascal Programming with an On-line Template Library and Case Sudies. *International Journal of Man-Machine Studies*, 20 (1993), pp. 1031-1048.

[CS73] W. C. Chase y H. Simon. Perception in Chess. *Cognitive Psychology*, 4 (1973), 55-81.

[ETW96] J. P. East, R. Thomas, E. Wallingford, W. Beck, J. Drake. Teaching Programming Through Patterning. *Proceedings of the 1996 Small College Computing Symposium*, April 1996, St. Cloud, Minnesota, pp. 39-45.

[GEH81] N. H. Gehani. Program Development by Stepwise Refinement and Related Topics. *BSTJ*, vol. 60, no. 3, March 1981.

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[GRI81] D. Gries. The Science of Programming. Springer-Verlag. Texts and Monographs in Computer Science, 1981.

[KNU84] D. E. Knuth. Literate Programming, *The Computer Journal*, 27, 1984, pp. 97-111.

[KP78] B. Kernighan, P. Plauger: The Elements of Programming Style. McGraw-Hill, 1978.

[PAT90] R. E. Pattis. A Philosophy and Example of CS-1 Programming Projects, *SIGCSE Bulletin*, vol. 22, no. 1, 1990, pp. 34-39.

[PRO00] V. K. Proulx. Programming Patterns and Design Patterns in the Introductory Computer Science Course, *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*,2000,80-84. http://www.ccs.neu.edu/teaching/EdGroup/

[SOL85] E. Soloway. From Problems to Programs Via Plans: The Content and Structure of Knowledge for Introductory Lisp Programming. *Journal of Educational Computing Research, 1(2)*, 1985, 157-172.

[SOL86] E. Soloway. Learning to Program = Learning to Construct Mechanisms and Explanations. *Communications of the ACM*, 29, 9 (Sept. 1986), 850-858.