Hacia una renovación en la enseñanza de la programación básica

Josep Maria Ribó

Departament d'Informàtica i E. Industrial
Universitat de Lleida
Plaça Víctor Siurana, 1
25006 Lleida
josepma@eup.udl.es

30 de mayo de 1998

Resumen

La enseñanza de la Programación básica en las titulaciones universitarias de informática sufre, a nuestro entender, un cierto anquilosamiento en cuanto a contenidos y herramientas. En este artículo presentamos los motivos en que basamos esta afirmación y proponemos una alternativa. El objetivo último no es el de dogmatizar sobre la docencia de la programación sino más bien el de participar en un debate que ya lleva mucho tiempo latente en la universidad española.

1 Introducción

En el ámbito de las titulaciones universitarias de informática no existe un amplio consenso sobre el enfoque que debe darse a la disciplina de la *programación básica*[1] más allá de ciertos elementos fundamentales como son las nociones algorítmicas, la programación estructurada o la introducción de las abstracciones funcional y de datos.

Por un lado se percibe que los nuevos paradigmas de programación que se han desarrollado en las últimas décadas no se han adoptado masivamente en los curricula de las disciplinas de programación básica. Por otro lado, es también patente la dificultad de encontrar un punto de equilibrio entre los defensores de una línea basada fundamentalmente en los métodos formales y aquellos de carácter mucho más practicista que tildan de contraproducente un excesivo celo en los aspectos formales de la programación. Finalmente tampoco parece haberse encontrado un lenguaje de programación universalemente reconocido como *adecuado* para introducir a los estudiantes en esta disciplina.

En este contexto, el objetivo del trabajo que presentamos es hacer una propuesta detallada de temario para 22.5 créditos en programación básica (desarrollados a lo largo de los tres primeros cuatrimestres de una titulación de informática técnica o superior) que equilibre los distintos puntos de vista al tiempo que introduzca el paradigma de la programación orientada a objetos desde el principio de la formación del estudiante.

El ánimo de la ponencia no es, en modo alguno, hacer una propuesta que tenga la pretensión de ser definitiva en el ámbito de la programación básica en las titulaciones de informática sino más bien el de hacer una aportación constructiva que pueda servir como elemento de discusión en el marco de este congreso o de futuros foros de discusión.

Este artículo se estructura del modo siguiente: En la sección 2 se hace una valoración global crítica del estado actual de la enseñanza de la programación básica en las titulaciones univer-

[1] En este trabajo englobamos dentro del término de *disciplinas de programación básica* aquellas que forman parte de los curricula de las distintas ingenierías en informática y que tienen como objeto la presentación a los estudiantes de las herramientas y metodologías fundamentales de la programación. Estas disciplinas suelen situarse, reseñadas con distintos nombres, a lo largo de los tres primeros cuatrimestres de dichas titulaciones y suelen tener una actividad lectiva de entre 12 y 15 créditos, el primer curso académico y entre 6 y 9, el segundo.

sitarias de informàtica españolas. La sección 3 presenta propiamente nuestra propuesta y constituye, por tanto, el nucleo de este trabajo. En la sección 4 se presentan brevemente las conclusiones de la misma.

2 La enseñanza de la programación básica. Situación actual

2.1 La situación actual

Si bien es cierto que existe una disparidad notable de criterios en cuanto a los elementos específicos de las materias que componen la *programación básica*, también lo es que se ha llegado a un cierto consenso respecto los contenidos fundamentales que deben impartirse en dichas materias. A grandes rasgos, estos contenidos podrían ser los siguientes: Especificación de problemas; diseño de algoritmos iterativos y recursivos; técnicas de programación estructurada; abstracción funcional y de datos; y presentación de las estructuras fundamentales de datos.

Estas grandes líneas argumentales son desgranadas de forma diversa en distintos centros, pero en un buen número de casos es posible apreciar también algunas coincidencias metodológicas[2]. Veamos algunos ejemplos:

- Presentación del diseño descendente como técnica para abordar el diseño de programas de tamaño pequeño-mediano ([3], [7], [8], [10], [11]).
- Uso de las secuencias de caracteres como dominio para el desarrollo de ejemplos y problemas ([3], [7], [11]).
- Presentación de los *tipos abstractos de datos* (TAD) como herramienta *formal* de abstracción de datos ([3], [7], [9], [10], [11]).
- Uso de los TADs para la descripción y *encapsulamiento* de las distintas estructuras fundamentales de datos ([3], [7], [9], [10], [11]).
- Utilización de algún lenguaje estructurado de reconocida validez y ampliamente experimentado en la docencia universitaria de progamación (Modula-2, Pascal, Ada...) ([3], [8], [9], [10], [11]).

Además de estos elementos metodológicos ampliamente extendidos, existen al menos otros dos, de carácter menos unánime aunque, creemos, ciertamente significativos. Éstos son, por un lado el elevado rigor formal con que estas disciplinas son abordadas en algunas universidades (pensamos que [3] es el ejemplo más paradigmático, pero también [9], [11]...) y, por otro, el orden de exposición de los contenidos que, en algunos casos, lleva a acabar el primer cuatrimestre con la resolución de problemas de una cierta complejidad (mediante la técnica del diseño descendente) para volver en el segundo cuatrimestre a abordar problemas algorítmicos más básicos (tratados, eso sí, desde un punto de vista formal). Esto sucede en [3], [7], [11]... entre otras universidades.

Pensamos que algunos de los elementos que hemos enumerado y que habitualmente forman parte de los curricula de programación básica en las titulaciones de informática de las universidades españolas son discutibles. O, al menos, creemos que es posible aportar otros planteamientos, desde ópticas diferentes, que pueden dar lugar a visiones complementarias y ciertamente enriquecedoras de la formación básica de los estudiantes en programación.

En el resto de esta sección plantearemos algunas reflexiones críticas sobre los actuales curricula de programación; en la próxima presentaremos una propuesta alternativa.

2.2 Una reflexión crítica

Algunos de los aspectos metodológicos y de contenido que, a nuestro entender son susceptibles de ser revisados o, al menos discutidos, son los siguientes:

- *La utilización del diseño descendente como técnica para la resolución de problemas.*

  La ingeniería del *software* actual ha consolidado para la resolución de problemas la *abstracción de datos* frente a las tradicionales metodologías *top-down* (basadas en la *abstracción funcional*; en la cual se basa también el diseño descendente); ver, por ejemplo [6]. Por este motivo, pensamos que la

[2] Las fuentes consultadas han sido: [3], [7], [8], [9], [10] y [11].

introducción en primer curso de una metodología de programación basada en la abstracción de datos puede ser mucho más natural y adecuada que el uso del diseño descendente.

- *La no incorporación del paradigma de la programación orientada a objetos (POO) en los curricula de programación básica.*

  Acaso la forma más natural de introducir la abstracción de datos ya desde primer curso sea utilizando el paradigma de la POO (nosotros así lo creemos y así lo manifestamos en nuestra propuesta), pero esta es una afirmación discutible. Lo que nos parece más difícil de justificar es que en la materia en que se introducen las estructuras de datos (habitualmente en el tercer cuatrimestre), no se utilice (ni se mencione tan siquiera) este paradigma. A nuestro entender, el trabajo con estructuras de datos parece diseñado expresamente para poder ser abordado desde la óptica de la POO: abstracción proporcionada por la clase y ocultación de los detalles de implementación; uso de clases genéricas para definir las estructuras de datos; uso de las jerarquías de clases y de herencia como herramientas para permitir la derivación de nuevas estructuras de datos; utilizaciones diversas del polimorfismo...

  La mayoria de los programas que hemos revisado de la materia de introducción a las estructuras de datos no se plantean presentarlas al abrigo de este paradigma. Debemos decir que este hecho nos parece un tanto insólito.

  En resumen, diríamos que nos parece un contrasentido que el avance claro hacia la abstracción de datos que se está produciendo en los últimos años en el campo de la ingeniería del *software* no encuentre una respuesta más rápida en los temarios de las asignaturas de programación básica, los cuales no parecen haber cambiado sustancialmente en los últimos decenios.

- *La utilización de lenguajes de programación de difusión nula fuera del ámbito docente.*

  Entendemos perfectamente (y compartimos) el argumento de la necesidad de que el primer contacto de un estudiante con la programación sea a través de un lenguaje *no traumático*. Este argumento (junto con la experiencia que algunos hemos cosechado con la enseñanza de programación en el lenguaje *C*) hace aconsejable el uso de lenguajes claramente estructurados, amigables, con un paso de parámetros razonable, con detección de errores en tiempo de ejecución, en definitiva, *de alto nivel*. Lenguajes como Pascal, Modula-2 o Ada cumplen estos requisitos. Pero también cumplen dos más: Son de uso prácticamente nulo tanto en el resto de las asignaturas de la titulación como en el mundo empresarial y no son orientados a objetos[3].

  Opinamos que es posible, e incluso aconsejable, utilizar otros lenguajes de uso más difundido, que soportan el paradigma de la POO y que cumplen los requisitos de elegancia y alto nivel preceptivos para la enseñanza de la programación. Opinamos que dos de estos lenguajes son C++ y Java.

  C++ y Java no son, en modo alguno, lenguajes *perfectos*. Por ejemplo, Java dispone de un mecanismo de paso de parámetros poco intuitivo para estudiantes que se están introduciendo en la disciplina de la programación. C++ tiene algunos problemas que discutimos en la sección 3. Sin embargo pensamos que son una buena apuesta para la enseñanza de la programación. Reafirman esta idea la nutrida bibliografía docente que ha aparecido en los últimos años en esta dirección.

- *El orden discutible de presentación de los contenidos*

  Ya hemos comentado anteriormente que éste no es un aspecto general que se produzca en gran parte de los curricula examinados pero sí en algunos que nos parecen significativos.

  A nuestro entender, los problemas de programación más complejos que se abordan en el primer curso de las titulaciones de informática (habitualmente problemas de tratamiento de secuencias que requieren una

[3] Excepto Ada, que en su última definición (Ada-95) sí puede recibir este apelativo.

cierta abstracción) deberían acometerse al final del curso, como consecuencia lógica de todas las técnicas aprendidas a lo largo del mismo. No parece tener mucho sentido plantear estos problemas al final del primer cuatrimestre y, en el segundo, volver a insistir en cuestiones algorítmicas que ya se plantearon anteriormente bajo una óptica, ahora, más formal.

- *Elevado grado de formalismo en la presentación de los temas*

  Nuestra experiencia como profesores de programación básica en la UdL es casi amarga, en este sentido. Hemos planteado a lo largo de bastantes cursos académicos los métodos de derivación formal de algoritmos como una herramienta fundamental de ayuda para el diseño de algoritmos. Hemos querido que esta herramienta de ayuda fuese un elemento esencial para razonar sobre la corrección de los mismos; un pecio al que agarrarse cuando las cosas no marchan bien, cuando alguna sutileza impide desentrañar algún error escondido. Sin embargo, para la mayoría de nuestros estudiantes, los métodos formales son un absoluto fastidio que jamás llegan a comprender en su totalidad y, mucho menos a usar con agilidad. Un obstáculo inútil que debe franquearse para la superación de la asignatura. En muchos casos, los estudiantes elaboran intuitivamente el algoritmo propuesto (en algunos casos correctamente!) y sólo después se plantean cual es el invariante o la función de cota del bucle.

  No estamos sugiriendo la eliminación de los métodos formales. Seguimos pensando que un ingeniero informático debe conocer el concepto de *invariante*, o saber qué pasos deben verificarse para concluir que un algoritmo recursivo es correcto. Sin embargo no estamos seguros que basar un curso en llevar estos aspectos a sus últimas consecuencias sea un planteamiento muy útil para un futuro ingeniero en informática. Creemos que deberían revisarse estos criterios al menos en el caso de las ingenierías técnicas.

En la sección siguiente hacemos una propuesta que, en la medida de lo posible, trata de paliar los elementos negativos que (según nuestro criterio) hemos encontrado en los temarios de las asignaturas de programación básica y que en esta sección hemos analizado.

3 Una propuesta de enseñanza de programación básica

En esta sección proponemos una organización de contenidos para 22.5 créditos de programación básica distribuidos en tres cuatrimestres. Estos créditos habitualmente se estructuran en dos asignaturas cuatrimestrales consecutivas en primer curso (por ejemplo *iniciación a la programación* y *programación metódica*) y otra, que suele orientarse a la presentación de las estructuras de datos fundamentales, en segundo curso, tercer cuatrimestre (por ejemplo: *estructuras de datos*).

Nuestro objetivo es que la organización de contenidos que presentamos pueda paliar, al menos parcialmente, las deficiencias que, a nuestro entender, se pueden encontrar en los temarios actuales y que hemos enumerado en la sección anterior.

3.1 Primer cuatrimestre (7.5 cr.)

El objetivo de esta materia es el de introducir a los estudiantes los elementos algorítmicos esenciales así como los diseños iterativo y recursivo. Después de esta introducción el estudiante debería ser capaz de de diseñar algoritmos recursivos e iterativos sencillos y razonar sobre su corrección. No se abordan todavía las técnicas de abstracción que permitirán resolver problemas de enunciado más complejo.

Parte I. Diseño iterativo: 4 créditos

1. Definiciones iniciales.

   Algoritmo, programa, tipo de datos, variable, constante, expresión válida, estado, especificación de un algoritmo, corrección de un algoritmo.

2. Especificación de algoritmos.

   Ejemplos de especificación formal de ciertos algoritmos. Planteamiento de la dife-

rencia entre la especificación y la implementación.[4]

3. Diseño de algoritmos.

   Presentación formal de las instrucciones algorítmicas básicas: asignación, composición secuiencial, alternativa e iterativa. Primeros ejemplos de algoritmos verificados.

   Presentación de acciones y funciones como herramienta de abstracción. Parámetros (tipos y mecanismos de paso).

4. Tipos estructurados.

   Presentación del *vector* y el *registro* como tipos estructurados que pueden definirse inmediatamente en el lenguaje algorítmico/de programación. Ejemplos. Presentación del tipo *cadena de caracteres.*

5. Algunos algoritmos iterativos.

   - Esquemas de recorrido y búsqueda en una secuencia. (Previa definición de secuencia). Ejemplos.
   - Búsqueda dicotómica.
   - Bipartición de un vector.
   - Inserción ordenada en un vector.
   - Algoritmo de ordenación de la *burbuja.*

   Estos algoritmos se derivarán formalmente. Pero su derivación irá guiada en todo momento por la intuición.

Parte II. Diseño recursivo: 3.5 créditos

1. Fundamentos de la recursividad.

   Presentación de la recursividad. Razonamiento formal sobre la corrección de las acciones recursivas. Ejemplos diversos.

2. Recursividad múltiple.

   Árbol de llamadas recursivas y peligros de esta recursividad. Diversos ejemplos.

3. Inmersiones.

   Inmersiones para mejorar la eficiencia de una acción recursiva, para lograr una acción recursiva final y para permitir el desarrollo de una acción recursiva (muy compleja de otro modo).

[4]Por supuesto no se realiza ningún diseño de algoritmos en este punto.

3.2 Segundo cuatrimestre (7.5 cr.)

Dos son los objetivos de este cuatrimestre: En primer lugar profundizar en los aspectos algorítmicos que se han introducido en el cuatrimestre anterior, presentando esquemas algorítmicos básicos, como el de *vuelta atrás* o el *divide_y_vencerás* y también algoritmos de ordenación clásicos como el *quicksort.* En segundo lugar presentar la abstracción de datos, y, en particular, el paradigma de la POO como estrategia para afrontar problemas de mayor grado de abstracción que dan lugar a soluciones que precisan de un volumen de código mayor.

Parte I. Algorítmica: 3 créditos

1. Eficiencia de los algoritmos.

   Introducción a la notación asintótica. Justificación intuitiva de la misma. Ejemplos de su uso.

2. Esquema de *vuelta_atrás.*

   Aplicación a los problemas de las ocho damas y la salida del laberinto.

3. Esquema de *Divide_y_vencerás.*

   Aplicación al problema de encontrar el máximo y el mínimo de una secuencia. Preparar los algoritmos de ordenación *quicksort* y *por fusión* como aplicación de este esquema.

4. Algoritmos de ordenación.

   Ordenación por inserción directa. Quicksort. Ordenación por fusión. Discusión de los costes.

5. Algoritmos voraces.

Parte II. Introducción a la POO: 4.5 créditos

1. Necesidad de los *tipos abstractos de datos.*

   Recordatorio del concepto de tipo de datos (como conjunto de valores + operaciones para tratarlos). Planteamiento de la necesidad de enriquecer los tipos disponibles en el lenguaje algorítmico (de programación) con *tipos abstractos* que permitan representar elementos del dominio del problema que se quiere resolver. Ejemplos. Presentación

de la abstracción como método para separar el comportamiento del tipo abstracto (definido en términos de operaciones convenientemente especificadas) y la representación/implementación de este tipo. Desarrollo de un ejemplo breve de definición, uso e implementación de un tipo abstracto.

2. Las *clases* y los *objetos.*

   Presentación de la clase como herramienta de la notación algorítmica (lenguaje de programación) para representar *tipos abstractos.* Definición de clases. Operaciones sobre clases. Separación entre *comportamiento* e *implementación* en las clases. Presentación de los objetos como instancias concretas de las clases.

3. Ejemplos de clases.

   Clase *Fecha*, *Complejo*, *Persona*, *Tablero_de_ajedrez.* Definición de las clases-ejemplo, especificación de sus operaciones, implementación de las mismas y uso.

4. Algunas características adicionales de las clases.

   Vectores de clases. Clases como atributos de otras clases. Operaciones constructoras.

5. La clase *secuencia_de_caracteres.*

   Definición. Uso. Esquemas algorítmicos usuales.

6. La clase *Fichero.*

   Definición. Uso.

7. Diseño de programas usando el paradigma de la POO.

   Planteamiento de problemas de tratamiento de secuencias de caracteres (16-20 horas lectivas).

3.3 Tercer cuatrimestre (7.5 cr.)

En este cuatrimestre se introducen las estructuras de datos bajo la óptica de la programación orientada a objetos. Por este motivo debe desarrollarse una tema inicial de repaso y ampliación de este paradigma. El resto se trata de un programa estándar de esta asignatura.

1. La POO.

   Abstracción. Encapsulamiento. Clases como herramienta para conseguir ambos. Objetos. Especificación de las clases. Excepciones. Jerarquías de clases. Herencia. Polimorfismo. Clases genéricas. Ejemplos abundantes.

2. Las estructuras de datos de acceso secuencial.

   Listas. Tipos de listas. Especificación de la clase *lista_con_elemento_distinguido.* Implementaciones de listas. Comparación de eficiencias. Listas con iteradores.

3. Las estructuras de datos de acceso directo: las tablas.

   Especificación de la clase *tabla.* Algunas implementaciones. La implementación mediante tablas de dispersión. Las relaciones. Ejemplos de diseño de estructuras de datos complejas.

4. Los árboles.

   Definiciones y propiedades básicas. Especificación de la clase *árbol_binario* y *árbol_general.* Implementación. Recorridos de árboles. La clase *cola_con_prioridad.* La clase árbol_binario_de_búsqueda. La clase *árbol-B.* Árboles con iteradores; implementación con enhebramientos.

5. Los grafos

   Definición y propiedades básicas. Especificación de la clase *grafo.* Estrategias de implementación. Distancias en grafos (algoritmos de Dijkstra y de Floyd). Árboles de expansión mínimos (algoritmos de Kruskal y Prim). Recorridos eulerianos y hamiltonianos en grafos (algoritmo de Hierholzer).

3.4 El lenguaje de programación C++

El lenguaje de programación que proponemos para el desarrollo de las tres asignaturas que hemos descrito es C++. Pensamos que C++ tiene una serie de ventajas sobre los lenguajes tradicionales que se utilizan en la docencia de programación básica:

- Es un lenguaje ampliamente usado en buena parte de las asignaturas de las titulaciones de informática[5] y en la mayoría de los ámbitos informáticos, con compiladores disponibles en todas las plataformas.
- Es un lenguaje de programación orientado a objetos que es el paradigma que se presenta desde primer curso en nuestra propuesta.
- Su utilización en la asignatura de introducción a las estructuras de datos (tercer cuatrimestre) se está convirtiendo en un estándar. Hay una muy abundante bibliografía docente disponible ([2], [4], [12]...) además de la reedición de clásicos como [5] bajo la óptica de C++. También es remarcable que Booch lo elija en [1] para presentar la POO.

Creemos, asimismo, que C++ supera la mayoría de aspectos que hacían de C un lenguaje *traumático* para el aprendizaje de programación:

- C++ distingue dos tipos de paso de parámetros (por valor y por referencia) perfectamente identificados. Este aspecto es crucial puesto que hace que no sea necesaria la presentación de los *apuntadores* hasta que aparecen de forma natural durante el tercer cuatrimestre en el apartado de las implementaciones en memoria dinámica de ciertas estructuras de datos. Repetimos, *con C++ se puede presentar todo el contenido de los dos primeros cuatrimestres sin un solo apuntador.*
- La definición de nuevos tipos en C++ es inmediata y elegante con el uso de clases. En particular es posible definir de forma transparente a los estudiantes la clase *secuencia_caracteres* de forma que ellos sean clientes de la misma.
- C++ hace comprobación estricta de tipos por defecto.
- La gestión de E/S es muy elegante. Define operadores de entrada (>>) y de salida (<<) universales. Estos son fácilmente sobrecargables.
- Implementa la práctica totalidad de los elementos del paradigma de la POO (exceptuando *persistencia* y *metaclases*). De hecho, tal vez sea C++ el lenguaje orientado a objetos más completo según el análisis comparativo de [1]. Esto lo hace especialmente interesante para trabajar en la asignatura de introducción a las estructuras de datos, en la cual el paradigma de la POO se estudia y utiliza en profundidad.

Desgraciadamente, C++ no resuelve adecuadamente algunos de los problemas que presentaba C: la definición de los vectores sigue siendo de bajo nivel y no se realiza ningún control en tiempo de ejecución de los accesos a vectores fuera de su rango de definición.

Sin embargo estos aspectos pueden ser resueltos fácilmente si enriquecemos C++ con los siguientes elementos:

- Definición de una clase genérica *vector* con control dinámico de acceso fuera de rango (vía excepciones) que sustituya los *array* de C. Con la creación de esta clase, la declaración de un vector de MAX elementos de tipo T se realizaría de la siguiente manera:

  ```
  vector<T> v(MAX);
  ```

  de forma consistente con las reglas declarativas de C++.
- Definición de un tipo *String* con las operaciones habituales de comparación de cadenas de caracteres, asignación...

  ```
  string s;
  ```

4 Conclusiones

En este trabajo hemos intentado sintetizar algunos aspectos que consideramos problemáticos de la docencia de la programación básica en las titulaciones universitarias de informática y hemos ofrecido una propuesta alternativa de contenidos para estas disciplinas. Los aspectos más relevantes de esta propuesta son los siguientes:

- Presentación del paradigma de la POO en primer curso.
- Profundización de este paradigma en segundo curso. Presentación de las estructuras de datos bajo con arreglo a él.

[5] más precisamente, C es un lenguaje ampliamente usado en este contexto.

- Reducción (de ningún modo eliminación) de la carga formal en primer curso.
- Uso de C++ como lenguaje de programación.

El objetivo último del mismo es el de ofrecer una propuesta que pueda servir como elemento de discusión entre aquellos profesores universitarios del ámbito de la informática preocupados por la docencia y, en particular, por la docencia de la programación básica.

Referencias

[1] G. Booch: *Análisis y diseño orientado a objetos con aplicaciones.* Addison-Wesley/Díaz de Santos, 1996.

[2] F.M. Carrano: *Data abstraction and problem solving with C++.* The Benjamin/Cummings Publishing Co., Inc., 1995.

[3] Facultad de Informática de Barcelona (UPC): *Guias docent.* Dirección: http//www-fib.upc.es/NovaGuia/etig.html.

[4] M.R. Headington, D.D: Riley: *Data abstraction and structures using C++.* D.C. Heath and Company, 1994.

[5] E. Horowitz, S. Sahni and D. Mehta: *Fundamentasl of data structures in C++.* W.H. Freeman and Co., 1995.

[6] B. Meyer: *Object-oriented Software Construction.* Prentice-Hall international, 1988.

[7] Guia docent de la UdL. Dirección: http://www.udl.es/rectorat/voa/index2.htm.

[8] Escuela Técnica Superior de Informática (UAM). Dirección: http://www.li.uam.es/esp/alumnos/index.html.

[9] Facultad de Informática (UPM). Dirección: http://www.fi.upm.es.

[10] Escola Tècnica Superior d'Enginyeria (URV). Dirección: http://www.urv.es/centres/estruc-cd.html.

[11] Centro Politécnico Superior (U. Zaragoza). Dirección: http://www.cps.unizar.es/spanish/acad/infor.html.

[12] M.A. Weiss: *Algorithms, data structures and problem solving with C++.* Addison-Wesley, 1996.