The Carnegie-Mellon Curriculum for Undergraduate Computer Science

Edited by **Mary Shaw**

pringer-Verlag
ew York Berlin Heidelberg Tokyo

TP3-43
S1

The Carnegie-Mellon Curriculum for Undergraduate Computer Science

Edited by
Mary Shaw

This curriculum and its description were developed during the period 1981–1984 by Stephen D. Brookes, Marc Donner, James Driscoll, Michael Mauldin, Randy Pausch, William L. Scherlis, Mary Shaw, and Alfred Z. Spector

Springer-Verlag
New York Berlin Heidelberg Tokyo

Mary Shaw
Stephen D. Brookes
Marc Donner
James Driscoll
Michael Mauldin
Randy Pausch
William L. Scherlis
Alfred Z. Spector

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213
U.S.A.

(C.R.) Computer Classification: K.3

Library of Congress Cataloging in Publication Data
Main entry under title:
The Carnegie-Mellon curriculum for undergraduate computer
science.
"This curriculum and its description were developed
during the period 1981–1984 by Stephen D. Brookes. . .
et al."
Bibliography: p.
Includes index.
1. Computers–Study and teaching. 2. Carnegie-Mellon
University. I. Shaw, Mary
QA76.27.C37 1984 001.64'07'11 84-23468

With 5 Illustrations

Printed and bound by R.R. Donnelley & Sons, Harrisonburg, Virginia.
Printed in the United States of America.

9 8 7 6 5 4 3 2 1

ISBN 0-387-96099-6 Springer-Verlag New York Berlin Heidelberg Tokyo
ISBN 3-540-96099-6 Springer-Verlag Berlin Heidelberg New York Tokyo

The Carnegie-Mellon Curriculum for Undergraduate Computer Science

Preface

Carnegie-Mellon has had a Computer Science Department and a PhD degree program since 1965, but an undergraduate major leading to a Bachelor's degree in Computer Science has never been offered. Nevertheless, a set of undergraduate courses is taught, and the Mathematics Department offers an option that relies heavily on these courses. Thus, an undergraduate student who wishes to study computer science will usually take a mathematics degree with a computer science option.

On a number of occasions over the past decade, the Computer Science Department has considered offering an undergraduate Computer Science major. Until recently, the decision has always been negative. In the Spring of 1981, however, the Department agreed to consider taking steps toward offering a major. A Curriculum Design Group was formed to identify the curriculum that would support a major. This group included Stephen D. Brookes, Marc Donner, James Driscoll, Michael Mauldin, Randy Pausch, William L. Scherlis, Mary Shaw, and Alfred Z. Spector. This book presents the result of the group's efforts.

We decided that the first step should be a thorough review of the existing curriculum. The content of the present courses has evolved through the years, and a complete review has not been done in quite some time. Because computer science is evolving rapidly, we felt that the changing nature of computers and computing was not adequately reflected by the existing curriculum. As a result, we decided to reconsider the entire curriculum, including both computer science courses and courses that will probably be offered by other departments. Our goals are described in a technical report [104] (reprinted in [103]) and briefly reviewed in Chapter 4.

At the same time as this design was underway, Carnegie-Mellon was developing a university-wide personal computer network. There was no substantial interaction between the curriculum design and the campus network project that was being designed concurrently, but we tried to identify ways to take advantage of advanced computing technology as we developed courses. In general, however, we believe that Computer Science Departments should coordinate their plans with those of their universities, which have a growing need to use computers in support of undergraduate education and to develop courses that deal with computation in fields other than computer science. We hope that by systematically including software support requirements in course designs we can influence the development of university computing facilities and justify software development as an ordinary part of course development.

Acknowledgments

A curriculum necessarily spans its discipline; its designers need all the help they can get. We have received a great deal. Though it is impossible to acknowledge all of it, we want to express our appreciation to some of the people who have affected our thinking most significantly. Thanks, then:

- To the Carnegie-Mellon Computer Science Department for support and encouragement in this project.
- To the following people, who contributed significantly to the design of individual courses: Jon Bentley, Ellen Borison, Jaime Carbonell, Wes Clark, Carl Ebeling, Allan Fisher, Ed Frank, Geoff Hinton, Takeo Kanade, Elaine Kant, Monica Lam, Dan Leivant, Matt Mason, Dana Scott, John Shen, Chris Stephenson, Hank Walker, Bob Wedig, and Bill Wulf.
- To Jill Fain, Cynthia Hibbard, Allen Newell, and Steve Shafer, who provided extensive critical comments on drafts of the proposal.
- To previous members of this project team, Jon Bentley and Guy Steele. Their participation in the early stages of the design contributed enormously to the final philosophy and structure.
- To members of the IEEE/ACM Software Engineering Planning Group. Section 2.2 was prepared while its author was working on both reports, and many of the ideas were developed or refined during the discussions at the planning meeting in September 1982.
- To the participants of the Sloan Conference/Workshop on the first two years of College Mathematics, from which the first version of our discrete mathematics syllabus emerged.
- To Roy Ogawa and Dana Scott for helpful comments on the original manuscript of chapter 8.
- To Betsy Grgurich, who typed, edited, and reformatted many, many versions of the manuscript.

Table of Contents

1. Introduction and Overview **1**
- 1.1. Goals of the Curriculum Design 1
- 1.2. Educational Philosophy 2
- 1.3. Character of the Curriculum 2
- 1.4. Innovations in the Curriculum 3
- 1.5. Organization of the Book 4

2. The Nature of Computer Science **7**
- 2.1. Working Definition of Computer Science 7
- 2.2. A View of Future Computing 8

3. Roles for Universities **13**
- 3.1. The Audience 13
- 3.2. Use of Computing Technology in Education 17
- 3.3. The ACM and IEEE Curricula 18

4. Objectives for the Curriculum **21**
- 4.1. Premises 21
- 4.2. Goals 21

5. The Content of Computer Science **23**
- 5.1. Basics 25
  - 5.1.1. Content 25
  - 5.1.2. Skills 26
- 5.2. Elementary Computer Science 26
  - 5.2.1. Content 27
  - 5.2.2. Modes of Thought 30
  - 5.2.3. Skills 31
- 5.3. Liberal Professional Education 31
  - 5.3.1. General Scope 31
  - 5.3.2. Liberal Education 31
  - 5.3.3. Areas Related to Computer Science 32
    - 5.3.3.1. Mathematics and Statistics 32
    - 5.3.3.2. Electrical Engineering 32
    - 5.3.3.3. Physics 32
    - 5.3.3.4. Psychology 33
    - 5.3.3.5. Mechanical Engineering 33
    - 5.3.3.6. Management and Information Science 33
    - 5.3.3.7. Public policy 33
- 5.4. Advanced Computer Science 33
  - 5.4.1. Control 34
  - 5.4.2. Data 34

5.4.3. Systems 35
5.4.4. Language 35
5.4.5. Foundations 35
5.4.6. Design 35
5.4.7. Communications 35
5.4.8. Applications 36

6. Program Organization 37

6.1. Requirements 37
6.2. Advice on the Use of Electives 40
6.3. Program Flexibility 41
6.4. Sample Program 42

7. Curriculum '78—Is Computer Science Really that Unmathematical? 45

7.1. Curriculum '78 and Mathematics 46
7.2. Mathematics for Computer Scientists 48

8. Mathematics Curriculum and the Needs of Computer Science 53

8.1. Some Words about Computer Science 53
8.2. Mathematical Aspects of Undergraduate Computer Science 54
8.2.1. Mathematical Modes of Thought Used by Computer Scientists 55
8.2.1.1. Abstraction and Realization 55
8.2.1.2. Problem-solving 57
8.2.2. Discrete Mathematics 57
8.2.3. Continuous Mathematics 58
8.3. Some Remarks about Computer Science and Mathematics Curricula 58
8.4. Conclusion 59

9. Theory and Practice in the Fundamental Computer Science Course 61

9.1. Introduction 61
9.2. Course Overview 63
9.3. Major topics in the Fundamental Structures Course 64
9.3.1. Models of Computation: Automata 65
9.3.2. Formal Languages 65
9.3.3. Formal Specification and Verification 66
9.3.4. Algorithmic Analysis 67
9.3.5. Data Types 67
9.3.6. Recursion 68
9.3.7. Programming Exercises 69

9.4. Experiences 69
9.5. Conclusions 70
10. Remarks on the Design 71
10.1. General Philosophy 71
10.2. Relation to Traditional Courses 72
10.3. Course Organization and Style 73
10.4. Course Numbering Scheme 74
11. Course Descriptions 75
11.1. Basic and Introductory Courses 77
11.1 1. Computers in Modern Society [100] 77
11.1.2. Programming and Problem Solving [110] 77
11.1.3. Discrete Mathematics [150] 79
11.2. Elementary and Intermediate Computer Science Courses 82
11.2.1. Fundamental Structures of Computer Science I [211] 83
11.2.2. Fundamental Structures of Computer Science II [212] 86
11.2.3. Real and Abstract Machines [240] 90
11.2.4. Solving Real Problems [300] 94
11.2.5. Time and Resources [310] 96
11.2.6. Program Organizations [313] 100
11.2.7. Languages, Interfaces, and their Processors [320] 103
11.2.8. Algorithms and Programs [330] 109
11.2.9. Formal Languages, Automata, and Complexity [350] 113
11.2.10. Logic for Computer Science [351] 117
11.2.11. Introduction to Artificial Intelligence [360] 119
11.2.12. Introduction to Robotics [361] 122
11.3. Advanced Computer Science Courses 124
11.3.1. Independent Project [400] 125
11.3.2. Undergraduate Thesis [401] 127
11.3.3. Research Seminar [409] 128
11.3.4. Big Data [410] 129
11.3.5. Communications and Networks [411] 132
11.3.6. Software Engineering [413] 135
11.3.7. Software Engineering Lab [414] 139
11.3.8. Transducers of Programs [420] 140
11.3.9. Advanced Programming Languages and Compilers [421] 143
11.3.10. Advanced Algorithms [430] 146
11.3.11. Computer Architecture [440] 148
11.3.12. VLSI Systems [441] 151
11.3.13. Theory of Programming Languages [450] 154

11.3.14. Complexity Theory [451] 157
11.3.15. Artificial Intelligence—Cognitive Processes [460] 159
11.3.16. Artificial Intelligence—Robotics [461] 162
11.3.17. Interactive Graphics Techniques [470] 164

12. Related Courses 167

12.1. Mathematics Courses 167
12.1.1. Introduction to Discrete Mathematics [Math 127 / CS 150] 167
12.1.2. Calculus I [Math 121] 167
12.1.3. Calculus II [Math 122] 167
12.1.4. Methods of Applied Math I [Math 259] 167
12.1.5. Elements of Analysis [Math 261] 168
12.1.6. Operations Research I [Math 292] 168
12.1.7. Operations Research II [Math 293] 168
12.1.8. Combinatorial Analysis [Math 301 / CS 251] 168
12.1.9. Linear Algebra [Math 341] 168
12.1.10. Numerical Methods [Math 369 / CS 352] 168
12.1.11. Modern Algebra [Math 473 / CS 452] 169
12.1.12. Applied Graph Theory [Math 484 / CS 430] 169
12.1.13. Numerical Mathematics I and II [Math 704 and 705] 169
12.1.14. Large-Scale Scientific Computing [Math 712 / CS 453] 169
12.2. Statistics Courses 169
12.2.1. Probability and Applied Statistics for Physical Science and Engineering I [Stat 211 / CS 250] 169
12.2.2. Probability and Statistics I [Stat 215] 170
12.2.3. Statistical Methods for Data Analysis I [Stat 219] 170
12.3. Electrical Engineering Courses 170
12.3.1. Linear Circuits: [EE 101 / CS 241] 170
12.3.2. Electronic Circuits I [EE 102 / CS 242] 171
12.3.3. Introduction to Digital Systems [EE 133] 171
12.3.4. Linear Systems Analysis [EE 218] 171
12.3.5. Electronic Circuits II [EE 221 / CS 340] 171
12.3.6. Analysis and Design of Digital Circuits [EE 222 / CS 341] 171
12.3.7. Introduction to Solid State Electronics [EE 236] 172
12.3.8. Introduction to Computer Architecture [EE 247 / CS 440] 172
12.3.9. Fundamentals of Control [EE 301] 172
12.4. Psychology Courses 173
12.4.1. Psychology of Learning and Problem Solving [Psy 113] 173

12.4.2. Information Processing Psychology and Artificial Intelligence [Psy 213] 173
12.4.3. Human Factors [Psy 363] 173
12.4.4. Cognitive Processes and Problem Solving [Psy 411] 173
12.4.5. Thinking [Psy 417] 174
12.5. Engineering and Public Policy Courses 174
12.5.1. Law and Technology [EPP 321] 174
12.5.2. Telecommunications Policy Analysis [EPP 402] 174
12.5.3. Policy Issues in Computing [EPP 380 / CS 380] 174
12.6. Engineering Courses 175
12.6.1. Real Time Computing in the Laboratory [Engr 252] 175
12.6.2. Analysis, Synthesis and Evaluation [Engr 300] 175
12.6.3. The History and Formulation of Research and Development Policy [Engr 401] 175
12.6.4. Cost-Benefit Analysis [Engr 404] 175

Index **187**

1. Introduction and Overview

The Carnegie-Mellon Computer Science Department's Curriculum Design Project examined the current state of computer science and computer science curricula, projected the requirements for undergraduate education in computer science over the next decade, and developed a curriculum suitable for a computer science major. This book presents the resulting design.

For many years Carnegie-Mellon has had a computer science curriculum (a body of courses), but it has never had a computer science major (a formal degree program). The department has adopted the curriculum presented here as a basis for major reorganization of its undergraduate offerings, though resource limitations will probably prevent a complete implementation of the curriculum. Some reasonable subset of the curriculum could form the basis for a computer science major. However, a curriculum is a necessary but not a sufficient condition for a major and the other issues, such as resource requirements, are not addressed here.

1.1. Goals of the Curriculum Design

Computer science is opening new specialties in many fields, and as a result the pattern of student involvement in university computing is changing. During the next decade, four different undergraduate populations within the university will require distinct kinds of education about computer science. These groups are:

- The *computer scientists,* students actually majoring in computer science,
- The *computing specialists,* students in computational specializations within other disciplines,
- The *occasional programmers,* students who will write programs for personal use; and
- The *casual users,* students who will make only casual use of computers.

In this design project we took as our goal the design of a curriculum for the first group of students: those interested primarily in computer science. We have formulated a unified view of the discipline, identified a suitable collection of courses, and defined the content requirement for a major. We chose not to consider the university resources required to support such a curriculum.

In this design, we did not address the computer science education of non-majors, but universities should do so. This report discusses the needs of these students and some suitable responses, but it does not go into depth. New curriculum designs will be required for the computer specialists and the casual users.

1.2. Educational Philosophy

We set out to develop a curriculum that would support a computer science degree of the highest quality. Such a curriculum requires a balanced blend of fundamental conceptual material and examples drawn from the best of current practice. In many ways, our educational philosophy is based directly on the Carnegie Plan for education [21, 22, 32, 88], which emphasizes an integrated understanding of basic concepts and the application of those concepts to practical problems. We believe that a curriculum with a small common core and a broad selection of advanced courses supports a variety of computer science specializations including both terminal and nonterminal programs.

1.3. Character of the Curriculum

We have designed a computer science curriculum consistent with this educational philosophy. The curriculum includes a unified overview of computer science, the content requirements for a computer science major, and detailed descriptions of a number of computer science courses. The curriculum has interactions with offerings in other departments, but these relations are not completely specified.

The design recognizes that computer science is a maturing field with a growing set of increasingly comprehensive models and theories. As such, it relies very heavily on mathematics, and it has close ties to several other disciplines. Because the field is changing rapidly, students need fundamental knowledge that they can adapt to new situations. In addition, students must be able to apply their knowledge to real problems, and they must be able to generate tasteful and cost-effective solutions to these problems. In this curriculum, the integration of theory and practice is a theme of virtually every course.

We have sketched outlines for thirty computer science courses. They include seven courses in systems and design, three courses in programming languages, two courses in algorithms and analysis, three courses in computer hardware systems, one course in elementary discrete mathematics, four courses in theory and mathematical foundations, four courses in artificial intelligence, one course in graphics, and five independent study, project, or seminar courses. Many of these courses are completely new, and the rest are revised from the form in which they currently exist at Carnegie-Mellon. As a result, a major effort will be required to implement the individual course designs.

In addition to the courses we define here, we have identified a number of courses generally offered by other departments that present material relevant to computer science. Though such material is often conceptually part of a computer science education, we did not develop new descriptions for such courses.

We have also proposed requirements for a computer science major based on this curriculum. These requirements are the basis for a liberal professional education. The required core is small (five specific courses plus two courses constrained to specific areas), thereby allowing a variety of specializations within the major. Additional requirements assure breadth, both by requiring substantial exposure to humanities, social sciences, and fine arts and by requiring a concentration of study outside the major.

1.4. Innovations in the Curriculum

Because the design was carried out without prior commitment to course organizations, the resulting organization is based on the structure of modern computer science rather than on traditional course divisions. The major innovative characteristics of the resulting curriculum include the following:

- *Organization around a core.* The curriculum consists of a core of courses that present the basis of the field together with a set of more advanced courses that provide depth of knowledge. The core courses emphasize the mathematical foundations of the field in practical settings.
- *Curriculum integration.* The courses are carefully integrated with each other, and strong prerequisite relations ensure that the material will be presented in a coherent order. Subareas often have one course that provides a broad introduction and a sequence of courses that provide greater depth.
- *Courses designed around ideas rather than artifacts.* Topics based on common ideas often appear in a single course, even if the topics are not traditionally taught together. This often entails rearrangement of traditional course boundaries; it also allows integration of theory and practice.
- *Use of proper computer support.* Many courses require extensive access to computers and software to illustrate points being made in the course. Though the forthcoming campus-wide personal computer system at Carnegie-Mellon will aid in this, we have presented the functional requirements for computer support rather than discussing specific ways to use personal computers.

After developing a global view of the curriculum, we derived specific courses. We re-derived the need for an elementary sequence much like the one developed at Carnegie-Mellon in the late 1970's (211 and 212)[1] This provides a solid foundation for sequences of advanced courses. In many cases, the initial course of an advanced sequence is eminently suited for a

[1]Numbers in this section refer to course numbers. Syllabi for these courses appear in Chapter 11.

student who wants to use the techniques of an area without specializing in it. The major new courses and course sequences include

- A sophomore course that provides a concrete appreciation of the nature of computation through a unified view of hardware, software, and theory (240).
- A reorganization of the traditional operating systems course that integrates the hardware, software, and theoretical views of concurrency, generalizes the resource management aspect of operating systems, deals with complex, long-lived data and integrates hardware and software aspects of communication (310, 410, 411).
- A new course that presents module-level program organizations and software development techniques. This course fills a gap between the courses that teach data structures or program fragments and the courses that deal with constructing systems from modules (313).
- A reorganization of the traditional comparative programming languages and compiler construction courses. The resulting courses focus first on programming languages and user interfaces, progresses to the use of advanced software tools for system (especially compiler) development, and culminate in language design and compiler construction techniques (320, 420, 421).
- A set of courses that present algorithms and the mathematical foundations of computer science with emphasis on integrating the practical aspects of the material with the presentation of the theory. The courses cover algorithms, logic, formal languages, automata, computability, complexity, and theory of programming languages (330, 350, 351, 430, 450, 451).
- A set of artificial intelligence courses that establish parallel sequences for the cognitive processing and robotics aspects of AI (360,361,460,461).

In addition, we plan joint development of a course for advanced students that establishes a basis for responsible evaluation of the consequences of computing and for interpreting the technology to laymen (380).

1.5. Organization of the Book

The setting for this design is discussed in Chapter 2. Roles for the university to play in the education of both majors and non-majors are examined in Chapter 3. Our general educational philosophy is defined in more detail in Chapter 4. Chapter 5 presents our integrated view of the content of computer science. Chapter 6 shows how majors (Bachelor's degree programs) could be created from the courses of this curriculum. Chapters 7 and 8 reprint articles on the role of mathematics in computer science and the nature of the curriculum support that computer science needs from mathematics. Chapter 9 discusses the design of the basic course, FUNDAMENTAL STRUCTURES OF COMPUTER SCIENCE I AND II [211/212]. Chapter 10 discusses the rationale for our organization. Outlines for the computer science courses we propose are presented in Chapter 11. Chapter 12 lists courses from other departments that cover related material.