A History of the Liberal Arts Computer Science Consortium and its Model Curricula

KIM B. BRUCE
Pomona College
ROBERT D. CUPPER
Allegheny College
and
ROBERT L. SCOT DRYSDALE
Dartmouth College

With the support of a grant from the Sloan Foundation, nine computer scientists from liberal arts colleges came together in October, 1984 to form the Liberal Arts Computer Science Consortium (LACS) and to create a model curriculum appropriate for liberal arts colleges. Over the years the membership has grown and changed, but the focus has remained on helping to establish and maintain high quality computer science programs in liberal arts colleges. In this report we discuss briefly the history of the group, the series of three curricula produced by LACS, and other contributions of the members to computer science education.

1. HISTORY OF THE LIBERAL ARTS COMPUTER SCIENCE CONSORTIUM
The Liberal Arts Computer Science Consortium (LACS) began in 1984 with a small group of dedicated educators from small liberal arts colleges working towards the goal of establishing a clearer definition of what computer science is and leveraging that definition into programs viable in the liberal arts landscape. These faculty examined the then current curricula standards as specified in Curriculum '68 [Atchison et al. 1968] and the more recent Curriculum '78 [Austing et al. 1979] and began looking for a more appropriate computer science curriculum for implementation in small liberal arts colleges. The group sought to capture the essence of the discipline as a core for a “model curriculum” practical for a department with a small faculty size and limited computational facilities (relative to the major research universities at the time).

Author’s address: K. Bruce, Computer Science Department, Pomona College, 185 E. Sixth St., Claremont, CA 91711.
Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2001 ACM 0000-0000/2001/0000-0001 $5.00
Specifically, the idea for the consortium evolved from a phone conversation between Norm Gibbs of Bowdoin College and Bob Cupper of Allegheny about some SIGCSE business that turned to a discussion of computer science at small colleges. The identification of a number of common problems, coupled with the suspicion that other colleagues in similar circumstances might be interested, led to the formation of a group created to provide a forum for the discussion and mutual solution of problems common to computer scientists at small liberal arts colleges.

Subsequently, Gibbs obtained a grant from the Sloan Foundation and hosted the first meeting at Bowdoin College in October of 1984. Nine faculty members\(^1\) came to that first meeting with lists of questions or problems to be solved:

—What kind of curriculum would be appropriate and realistic in the small liberal arts college environment?

—How could we attract faculty to this kind of environment?

—Could we offer a faculty exchange program? The exchange might facilitate research collaboration and perhaps make us more competitive in seeking research grants.

—Could we offer a student exchange program? Given the small number of faculty in each of the schools, such a program could provide a greater opportunity for breadth for our students than any of our small liberals arts colleges could offer alone.

—Could we define a reasonable basic set of laboratory facilities for computer science? How could we justify the need for laboratory facilities to colleagues and administrators who did not think of computer science as a laboratory science?

—Could we collaborate in seeking grants and/or discounts to implement labs in our institutions?

While the notions of faculty and student exchanges faded over time, and gradually a combination of increasing budgets and falling computer costs enabled departments to provide high quality laboratory facilities, the first two items have remained important issues over the last 25 years. The design of curricula, in particular, has been a regular focus for the LACS meeting attendees.

The “Model Curriculum” that evolved from that first meeting, and the subsequent one at Colgate in the summer of 1985, had a significant effect on our own programs, those of many other schools, and on the work of the Joint Curriculum Task Force that produced Curriculum ’91 [ACM/IEEE-CS Joint Curriculum Task Force 1991]. In addition, discussions among consortium members over the next several years in meetings at Allegheny in 1986, Williams in 1987, and Washington and Lee in 1988, focused on the question of the role of laboratories in CS introductory courses. This effort resulted in a report [Parker et al. 1990] describing both the formal dedicated laboratory concept and the concomitant required equipment and support for labs.

In the immediately succeeding years, the consortium members decided that time, as well as the appearance of Curriculum ’91, demanded that the “Model Curricu-

\(^1\)The faculty attending that first meeting were: Kim Bruce, Robert Cupper, Norman Gibbs, Stuart Hirshfield, Nancy Ide, Charles Kelemen, Jeffrey Parker, Ted Sjoerdmsa, and Allen Tucker.
lum” be reexamined. Much of the discussion focused on the center of the curriculum, the “core.” So much had changed that there was a need to add topics such as parallelism, concurrency, operating systems, networks and distributed processing as essential to a basic undergraduate education in computer science. After concluding that the number of core courses could not be extended beyond the original four, the group spent considerable time and effort deciding what topics needed to be added and which of the then current topics were at least in part expendable. Members also worked on the integration of a research component into the curriculum. The result was published as “The Revised Model Curriculum,” in 1996 [Walker and Schneider 1996]. A decade later, again motivated in part by the appearance of the ACM/IEEE CS Curriculum 2001, the third incarnation of the “model curriculum” appeared in 2007 as “A 2007 model curriculum for a liberal arts degree in computer science” [Liberal Arts Computer Science Consortium 2007].

Throughout, the group discussed, though never came to a public position on, matters of workload, problems of keeping pace with rapidly changing technologies and pedagogies, changing enrollment patterns, and the cyclic difficulties of recruiting and evaluating faculty.

From the very beginning, membership, organization, and size were issues. The Consortium began and remains to this day a small group of 15 to 20 individuals, each, to be sure, associated with a liberal arts college, but participating as individuals. The meetings were and continue to be in many respects more like meetings of a working group than a conference or workshop. Issues such as topics to be addressed, size, and membership are determined by group consensus at an informal meeting at the annual SIGCSE conference and by e-mail conversations. The group meets each summer, rotating not only our meeting place among the college campuses of our members, but also the positions of convener and secretary among the members.

The initial funding ran out rather quickly, but LACS members were able to create a simple funding model to support continuing annual summer meetings. Each member is responsible for finding funding to get to the airport nearest to the meeting location, while the host for the meeting provides funding for room, board, and other costs of supporting the meeting. Deans at member schools are generally happy to provide funding for meetings that provide such obvious benefits for faculty and the institutions.

Contributions of the Liberal Arts Computer Science Consortium to computer science education include:

—Developing an initial “model curriculum” [Gibbs and Tucker 1986] designed to be compatible with the liberal arts orientations of the participating schools.

—Maintaining currency in the “model curriculum” reflecting ongoing development and technical changes in computing [Walker and Schneider 1996; Liberal Arts Computer Science Consortium 2007].

—Pioneering the design and development of the laboratory component of the computer science curriculum [Parker et al. 1990].

—Providing justification for the importance of mathematical ideas in computer science [Bruce et al. 2003; Tucker et al. 2001; Kelemen et al. 2000].

—Providing a resource for the evaluation of computer science programs nationwide.
—Providing lecturer/workshop exchanges among members and their host institutions.
—Developing collaborative projects among members.
—Providing a forum for exchange of ideas and discussion of problems common to computer scientists at small liberal arts colleges.
—Developing research-oriented capstone experiences for the undergraduate degree in computer science.

The remainder of this paper chronicles the development and subsequent revisions of the Liberal Arts Model Curriculum. While we present an overview of each curriculum here, we refer readers to the original published versions [Gibbs and Tucker 1986; Walker and Schneider 1996; Liberal Arts Computer Science Consortium 2007] for details.

2. THE FIRST LACS CURRICULUM

The first LACS curriculum resulted primarily from the growth of interest in computer science at many small highly selective liberal arts colleges. However an added incentive was widespread dissatisfaction with existing curriculum standards in computer science and concerns about the establishment of an accreditation program in computer science. It is important to look briefly at these as they represented a tendency to move the study of computer science in a direction less amenable to liberal arts colleges.

In 1984 there had already been strong criticism of ACM Curriculum '78. There were two main concerns. First, Curriculum '78 segregated its mathematics requirements from the computer science courses, as none of the mathematics classes were prerequisites for core CS courses [Ralston and Shaw 1980]. In fact in an early meeting, the curriculum '78 committee agreed that “no mathematics, beyond high school was required for study of the heart [of CS]” [Engel 1974].

The second major complaint dealt with the content of the Curriculum '78 requirements. As stated in the first LACS curriculum report, “The core curriculum in computer science is frequently questioned because it seems to be composed of a collection of different programming and applications courses and fails to explicate adequately the principles that underlie the discipline” [Gibbs and Tucker 1986].

In the early 1980’s there was a push by some engineering-oriented CS departments and faculty to establish an accreditation program for computer science. Not surprisingly, the resulting Computer Science Accreditation Board had a strong bias toward a professional engineering-style education. As a result, the course specifications were more prescriptive and required more narrow concentration on computer science and mathematics courses than would be appropriate (or even allowed) at many liberal arts colleges.

These developments, along with the initial difficulties of selling computer science to relatively skeptical faculties and administrations at liberal arts colleges led to the desire by faculty from these colleges to develop a curriculum for computer science that would be more appropriate in the environment of the small liberal arts college.

To begin, it was necessary to present a definition of computer science [Gibbs and Tucker 1986]:
“Computer science is the systematic study of algorithms and data structures, specifically
(1) their formal properties,
(2) their mechanical and linguistic realizations, and
(3) their applications.”

The report noted that computer science programs should emphasize the formal properties over the other two aspects. Unfortunately, this led some readers to believe that this meant that the group believed that computer science should primarily be theoretical or mathematical, while the actual intent was that computer science should be taught based on important underlying principles, many of which were amenable to being formalized. Later revisions of the curriculum by LACS would attempt to clear up this misunderstanding.

2.1 The Introductory Courses

The recommendations for CS 1 were essentially the same as those presented by the ACM Curriculum Committee Task Force for CS1 [Koffman et al. 1984], while those for CS2 differed a bit from the corresponding task force for CS2 [Koffman et al. 1985] by suggesting that the course also provide more of an overview of computer science.

Extra topics recommended for CS2 were:

(1) unsolvable problems: brief treatment of the halting problem (in a high-level language);
(2) systems software: a brief survey of simple architecture and assembly language; discussion of assemblers, compilers, operating systems, etc., with emphasis on important algorithms and data structures (e.g., parse trees and symbol tables for compilers).

While seemingly ambitious, the report recommended only an overview of these topics with only one day spent on the halting problem and one to two weeks for the discussion of the principles of systems software. For the coverage of systems software the report recommended: “This should not be a series of lectures on the architecture and particular systems software of the local computer; instead, the idea is to use this survey to pull together many of the data structures . . . used in CS2.”

The report also called for consideration in CS2 of the usage of a procedural language such as Modula-2 or Ada for its support of data abstraction or, alternatively, of a functional language like LISP that departs dramatically from the procedural and object-oriented languages typical of CS 1 and CS 2 courses and illustrates alternative ways of thinking about the programming process.

2.2 Mathematics Requirements

The mathematics requirements in the LACS curriculum were designed to provide important preparation for core courses in computer science. The first and most important course was the study of discrete mathematics. This course was to contain an introduction to logical reasoning (including methods of proof, especially proof by
induction), counting and simple finite probability theory, and the rudiments of sets, functions, and relations. Other recommended topics included recurrence relations and difference equations, graphs and trees, and matrices. This course was designed to be a co- or prerequisite for CS2, so that these mathematical topics could be relied on in discussions of complexity and correctness.

The second required mathematics course was primarily designed to increase the student’s level of mathematical maturity. It could be a second discrete mathematics course, a calculus course, or a linear algebra course, depending on the preferences of the department and the needs of the particular student’s program of study. Students planning on going to graduate school were encouraged to take substantially more mathematics.

By way of contrast, Curriculum ’78 required six mathematics courses, but they were not listed as prerequisites for any of the required courses in the curriculum. Only the elective in theory of computation and two numerical analysis electives had mathematical requirements. Even the algorithms course had no mathematical prerequisites.

2.3 Core Requirements

Reflecting the definition of computer science given in the report, the core curriculum was focussed on principles, requiring four courses:

CO1. Principles of Computer Organization,
CO2. Algorithms,
CO3. Theory of Computation, and
CO4. Principles of Programming Languages.

The content of these core courses was fairly standard. However, there was a strong emphasis on principles underlying the topics. For example, the programming languages course was focussed more on principles than were older-style survey of languages courses. However, students were also expected to get some practice with new languages, especially those using then less common styles, e.g., functional, object-oriented, and logic programming languages.

The choice of these four courses as core contrasted with the six core courses of Curriculum ’78:

CS 3. Introduction to Computer Systems,
CS 4. Introduction to Computer Organization,
CS 5. Introduction to File Processing,
CS 6. Operating Systems and Computer Architecture I,
CS 7. Data Structures and Algorithm Analysis, and
CS 8. Organization of Programming Languages.

The greatest difference between these selections, aside from the size of the core, is the heavy emphasis on architecture and systems, as exemplified by the first four courses. By way of contrast, the theory core course (CO3) from the LACS curriculum appears only as an advanced elective (CS16) in Curriculum ’78.

Moreover the last two courses in the above list have considerably different emphases than the corresponding CO2 and CO4 from the LACS curriculum. The
Algorithms course in Curriculum '78, CS7, specified 25% of its lecture hours on integrating data structures into a simple database management system, while the specification of the programming languages course, CS 8, emphasized applied (especially programming) aspects, and read more like a survey than a principles course. As noted earlier, none of the introductory or core courses had any mathematics prerequisites.

2.4 Electives

The LACS curriculum required that each student take 3 electives in computer science, at least two of which should be selected from courses in compiler design, artificial intelligence, operating systems, database principles, or computer architecture. The LACS curriculum report also recommended that each student be required to complete a substantial software project, including a significant writing component in one of the electives.

By way of comparison, Curriculum '78 required 4 electives, with no more than two courses in any particular subfield of CS.

2.5 Summary of the 1986 LACS curriculum

Considering that only eight years had passed since the publication of Curriculum '78, the differences between the two curricula recommendations were considerable. Where Curriculum '78 required 18 courses, with 12 in CS and 6 in mathematics, the LACS curriculum required only 11 courses, 9 in CS and 2 in mathematics, reflecting the broader nature of the liberal arts degree.

However, this simple counting of required courses obscures more significant differences between the curricula. While the number of mathematics courses is smaller in the LACS curriculum, the mathematical and theoretical components of the LACS curriculum were much more significant than in Curriculum '78. This reflects the fact that an important goal of the LACS model curriculum was for students to gain considerable facility at applying mathematical and logical analyses to computing topics.

3. REVISIONS OF THE LACS CURRICULUM

There have been two revisions of the LACS curriculum.

3.1 Revised Model Curriculum of 1996

A new set of curricular recommendations were published by the ACM/IEEE Joint Curriculum Task Force in 1991 [ACM/IEEE-CS Joint Curriculum Task Force 1991]. Curriculum '91 differed in a number of ways from the Curriculum '68 and Curriculum '78, but the largest change was that it did not recommend a curriculum. Instead it broke the topics in the core of computer science into 56 “knowledge units” in 11 areas. It then presented a number of sample curricula, showing alternate ways of packaging those knowledge units into courses.

The members of LACS took Curriculum '91 as an opportunity to develop a revised model curriculum [Walker and Schneider 1996]. This curriculum showed a packaging of knowledge units in a way that made sense in a liberal arts setting. The group thought it best to have a single curriculum rather than a larger number of sample curricula. The revised curriculum also incorporated educational and
technological developments that had occurred in the previous ten years. LACS members and others had been experimenting with formal laboratories in introductory courses, breath-first curricula, and senior projects. Many of these ideas found their way into the revised curriculum.

The revised curriculum kept the idea of levels of courses from the ’86 model curriculum. It recommended 2 introductory courses (Computer Science I and Discrete Mathematics), 2 intermediate courses (Computer Science II and Computer Organization and Architecture), 3 core courses (Sequential and Parallel Algorithms, Foundations of Computing, and Programming Languages and Systems), and 3 elective courses. The intermediate level also included two additional mathematics courses that support the core courses and electives.

The revised curriculum devoted approximately 240 hours to the core topics in computer science in the introductory, intermediate, and core courses. This contrasted with 283 hours devoted to these topics in Curriculum ’91. Most of the difference in hours came in the areas of architecture and software engineering. However, in keeping with the liberal arts goal of emphasizing fundamental principles, the revised curriculum recommended almost 30 hours more on algorithms and data structures than Curriculum ’91. It was not a pared-down version of Curriculum ’91. It was a curriculum with a different emphasis.

The revised curriculum included a number of new suggestions about the first two courses. It asserted the importance of students seeing multiple paradigms in programming languages early in their studies. Different paradigms facilitate different ways of thinking about and organizing computation. Students who spend too long in a single paradigm can come to think of it as “the way” to program, and often have a difficult time learning other paradigms later. Therefore the revised curriculum recommended that students see at least two paradigms within their first two courses. These two were to be chosen from procedural, object-oriented, and functional paradigms.

The revised curriculum also recommended scheduled, formal laboratory periods for the first two courses. The advantages of formal labs were discussed in [Parker et al. 1990].

Finally, the revised curriculum explicitly recognized that the first few courses could be organized in a traditional way or the topics could be interwoven using a breadth-first approach. Instead of a first course that is mostly programming and later courses that bring in other aspects of computer science, the goal was to include a broader overview of computer science in the first course, swapping some of the programming topics that are in the traditional first course with other topics that usually appear in later courses. There was also a breadth-first sample curriculum in Curriculum ’91, so this was not the first curricular recommendation to deal with the breadth-first paradigm.

The revised curriculum introduced other changes: more coverage of parallelism and parallel algorithms, integrating theory throughout the curriculum instead of relegating it to one course, inclusion of social and ethical issues in low-level courses, an increase in the number of required mathematics courses, and the inclusion of a senior project.

The senior project had rather specific requirements. It included three compo-
ponents: reading one or more scholarly papers from the CS literature, writing a
significant paper or technical document, and making an oral presentation to stu-
dents and faculty. This project could be an independent research project (e.g.,
an “honors project”), a project in an elective course, a group software-engineering
project, a review of the literature in an area, or some other appropriate activity.
The combination of studying the literature and presenting the results both orally
and in written form fit well in the liberal-arts approach.

3.2 2007 Model Curriculum

In 2007 LACS produced a third model curriculum[Liberal Arts Computer Science
Consortium 2007]. This 2007 model curriculum considered the 2001 report of the
Joint Task Force on Computing Curricula[The Joint Task Force on Computing
Curricula 2001] and other developments that had occurred since 1996. Rather than
providing a detailed explanation of the contents of this most recent curriculum
recommendations, we simply provide a brief outline of the major changes and refer
the reader to the actual curriculum report.

One of the biggest changes was the addition of a new software development course
at the intermediate level. This addressed a perceived weakness in the earlier model
curricula. Students often learned about software development through program-
moving projects in elective courses and their senior projects, but had little formal
instruction in software design, design patterns, unit testing, APIs (GUIs, event-
driven programming, networking, etc.), and other software development topics.
This intermediate-level course provides a systematic study of software development
and allows students to do more extensive projects in the elective courses and the
senior project.

Adding this course required reorganizing material in other courses as we wished
to keep the core to roughly the same size as that of the earlier LACS curricula.
The new curriculum’s introductory level consists of Computer Science I, Computer
Science II, and Foundations of Computer Science I (which covers topics in discrete
mathematics). The core was expanded to five courses: Computer Organization,
Software Development, Algorithms, Programming Languages, and Foundations of
Computer Science II. This model requires three CS electives, but the number of
mathematics electives has been reduced from two to one, in an area to be selected
by the student in consultation with a faculty advisor.

A second decision was that the object-oriented paradigm and procedural paradigm
were too similar given the need to expose students to two different approaches
to thinking about and organizing computation. Thus the 2007 model curriculum
specifically recommends that students study both the object-oriented paradigm and
the functional paradigm during their first few courses. (Logic programming might
be substituted for functional programming in some schools.)

The 2007 model curriculum presents two ways of organizing CS I, CS II, and
Foundations of Computer Science (FC I). One does object-oriented programming
first. The other does functional programming first. Schools (including ones within
LACS) have used both approaches successfully, and members chose not to recom-
mend one approach over the other.

The objects-first approach recommends a fairly traditional CS I and CS II, but
uses functional programming to introduce and illustrate discrete mathematics topics
in FC. By way of contrast, the functional-first approach keeps FC as a traditional discrete mathematics course, but CS I is a course in functional programming and CS II is a course in object-oriented programming and data structures.

Foundations of Computing II is no longer the traditional Theory of Computation course. Automata and computability are studied, but grammars are moved to the Programming Languages course. The rest of FC II covers more advanced discrete mathematics topics (probability, statistics, and number theory). The main effect of this change is to better integrate discrete mathematics and theory in this course, while other theoretical material more closely related to programming languages is moved into that course.

4. CONCLUSION

For twenty-five years the Liberal Arts Computer Science Consortium has played an important role both for its members and for other computer science faculty at liberal arts colleges and other institutions with similar educational philosophies. Annual discussions among the fifteen to twenty active members help participants solve problems and suggest innovative pedagogy. Published reports on curriculum recommendations, scheduled laboratories, and other topics have helped an even wider group.

We believe that the success of LACS is related to the following features of the organization:

(1) The membership is intentionally kept small, with new members invited based on their abilities to contribute. When a member resigns or retires, new candidates from a variety of liberal arts colleges are considered. Because it is a membership of individuals rather than departments, there is not an automatic presumption that the replacement will come from the same department.

(2) The focus of the organization is clear: Share, develop, and publish ideas for increasing the quality of computer science programs at liberal arts colleges.

(3) Most of the costs of the meetings is borne by the host of each meeting, and the meetings rotate among the membership. Attendees are only responsible for getting to the airport nearest to the meeting site. Deans are usually willing to

---


3Three of the original members of LACS have since been presented the SIGCSE Award for Outstanding Contributions to Computer Science Education. Two members served on the Curricula ‘91 report (with one serving as ACM co-chair), while another served on Curriculum 2001. Many other members played important roles in developing these curricula by serving on task forces in support of particular subspecialties in Computer Science. Members have also served as officers and members of the board of SIGCSE, and have been involved in the organization of SIGCSE meetings.
contribute the relatively small amounts (typically less than $6000) necessary to fund the summer meetings on campus, with attendees residing in dormitories. Planning for summer meetings is done at short meetings during the SIGCSE Technical Symposium on Computer Science Education each spring.

The LACS membership looks forward to many more years of making contributions to computer science at liberal arts colleges.

REFERENCES


Received Month Year; revised Month Year; accepted Month Year