IMPLEMENTING A COMPUTER SCIENCE CURRICULUM MERGING TWO CURRICULUM MODELS

WILLIAM MITCHELL
BRUCE MABIS
UNIVERSITY OF EVANSVILLE

ABSTRACT.

While many curriculum recommendations have been proposed, (1,3,4,5,9), there remain gaps in the specification of 4-year programs, one being in the area of the applied B.S. degree. This paper presents one attempt to fill this gap with a program whose goal is to produce competent computer professionals, not candidates for graduate schools. Within the constraints of a limited faculty, the quarter calendar, and an applied orientation, it still incorporates much of the thrust and content of the more theoretically oriented recommendations. We believe that this effort may serve as a model for other schools who share our charge to train applications oriented professionals, and that the curriculum design process which we illustrate may be useful in even broader contexts.

HISTORY

The computing science program at the University of Evansville began in 1962 with a single course entitled "Principles of Data Processing", offered on the University's IBM 1620. Four years later a two-year degree was offered in the Evening School called Computer Technology with scientific (mathematics) or commercial emphasis. Ten courses, each 3 quarter hours credit, were available, ranging from "Introduction to EDP" through "Assembly Language Programming" and "COBOL" to "Systems Analysis I & II". In 1969 the University installed an IBM 360/20, and in 1970 the Evening School offered a B.S. degree in Computing Science. This degree required 32 quarter hours of computing science (upgraded courses from the associate program), 33 quarter hours of mathematics (calculus and numerical analysis), and 36 quarter hours of business. The University moved up to an IBM 370/135 in 1972, and the computing science program was transferred from the Evening School to the School of Arts and Science.

Growing concern for the appropriateness of the major requirements, both within and without the University, led in 1973 to the request by the Academic Vice President that the program be radically upgraded. In 1974 a new major was implemented which required 54 quarter hours in computing science through the addition of 8 new courses to the offerings. The senior level sequence in systems analysis was changed to a sequence in information and management information systems, and the mathematics requirement was cut by two-thirds. The business component became the usual choice for a "related area" of 40 quarter hours. Although the new courses included introductory architecture, computer electronics, compiler design and digital logic, the required courses of the major continued to be heavily language oriented. A third faculty member was added to the department in 1975 to make the additional offerings available. The curriculum revision was formulated with scant attention to the ACM 1968 recommendations (5), which were seen as irrelevant to the objectives and commercial orientation of the program.

In 1976 the program underwent a second crisis when the Academic Vice President transferred the program to the School of Engineering and Applied Science. This resulted in the eventual resignation of the two senior members of the department, and provoked a systematic evaluation of the program by their replacements. Under the direction of the Dean of the School of Engineering and Applied Science and an advisory committee appointed by the President, the department produced a new curriculum. Both department members were active in SIGCSE and had for guidance the ACM Curriculum'78 recommendations (3) and the IEEE Computer Science and Engineering curriculum report(9). What follows is the account of how the curriculum was built upon these models within the restraints of available staff, program orientation, and the quarter calendar.

ASSESSMENT

The first step was an analysis of how the current offerings compared with the ACM recommendations. In terms of course titles, there was a wide variance, but it was concluded that many of the topics suggested for the core recommendations were covered to some degree. Clearly the existing offerings were not oriented toward graduate preparation, nor did they require mathematical competence beyond algebra. There was a partial overlap with topics in the Information System recommendations of 1973 (4). The most serious weaknesses were in the introductory sequence (which contained very little programming) and the absence of formal treatment of file and data structures.

The second step was to consider the opportunities of the program to interface with other departments. The School of Business already required the introductory course in BASIC programming

which was offered as a service course to the University. They were unhappy, however, with the low level of programming experience provided due to the time devoted to surveying historical and societal phenomena. The department of Electrical Engineering, on the other hand, had formulated a major in Computer Engineering with an emphasis on microcomputers. It encouraged its majors to take several upper division computing science courses while skipping their prerequisites. Despite the maturity of these students and their excellent mathematical preparation, they lacked the practical experience of getting several programs completed in a quarter. The other departments in the University which might be expected to encourage students to seek a minor or at least a programming proficiency in some language were discouraged by the fact that all language instruction had a full year of prerequisite data processing courses (both the School of Engineering and the Mathematics Department offer FORTRAN courses for their own students). Clearly better access to the computing science courses was needed, with minimum but enforceable prerequisites.

The third step in our assessment was to consider the nature of the major itself and the students it was intended to serve. In spite of the recent relocation, the students in the forseeable future would continue to be commercially oriented and continue to be employed in the immediate area after graduation. What specific characteristics were desired for these students? Since they were training to be computer professionals, they should be skilled language users, practiced in all aspects of their craft, and familiar with the role they would occupy in the corporate effort. They should be like compent and versatile musicians, understanding the mechanics of their instrument, the circumstances under which they perform, and the repertoire which will be expected.

Our final constraint centered on program resources. We enjoyed the support of the Engineering School, but it was not prepared to subsidize program expansion. Current student enrollment could not justify increasing the full time computing science faculty beyond three. Since we could continue to employ a full-time equivalent in part time help (in service areas), the curriculum had to be structured in 144 quarter contact hours per academic year. The new curriculum had to be reasonably coherent with the existing program in order to accommodate the transition period, and it could not require significantly more computer resources than the existing program. Insofar as our curriculum could better serve electrical engineering, we could expect instructional assistance in the cross listed courses. We had no similar arrangement with the School of Business. We also had no expectation of any graduate level offerings. These limitations were subject to change, and could conceivably change rapidly in response to a spurt in enrollment, but they were ever present in the background as we began the redesign process.

DESIGN

Following the Engles and Austing "stages" curriculum model (2), we divided the proposed offerings into the categories of service offerings, major core and major electives. Service courses were divided into general education offerings which would not count toward the major and other offerings which would be taken by majors but taught in a manner to attract science and business minors. The major core was divided among three areas of emphasis, similar to the IEEE design: systems, applications programming, and computer hardware and software. The superposition of the two category systems resulted in the model specified in figure 1.

This diagram suggests several things we feel are true about our program. It makes evident our interfaces with the School of Business, the Department of Electrical Engineering, and the University in general. In the center, as is appropriate for us, is the emphasis on programming competence. The spread of the major core through all three areas insures breadth and balance in our students. The overlap of the major core and the service area reflects not only the scarcity of our faculty resources, but speaks to our goal of providing a broad range of service offerings to the University and thereby increasing our visibility. Thus, out of a total of 122 quarter hours of offerings, one third are in the service area and may be taken with (at most) one prerequisites.

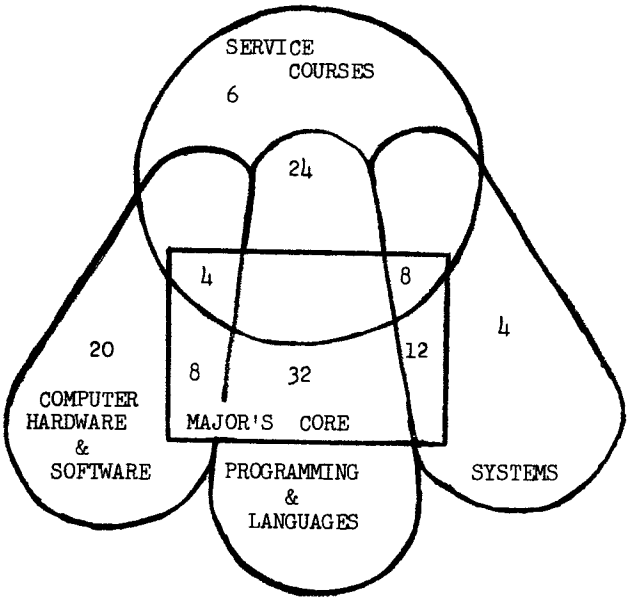

Figure 1. Computing Science Curriculum Category Structure. Numbers are the total hours available in each region (for example, of the 24 hours in the systems area, 20 are part of the major's care, and 8 of those are also service oriented.

Since the major's core is the backbone of a degree which serves a variety of professional orientations, the content of that core was carefully chosen and organized. Particularly important was the recognition that a significant portion of our students are unsure of their interests and capabilities when they enroll, and that there is a fundamental difference between amateur and professional programming, whatever similarities might also exist.

We designed the core to include the content of ACM '78 curriculum's 8 core courses (equivalent to 36 quarter hours). Beyond that material we have 20 quarter hours in the systems area, emphasized more heavily by IEEE, including the 8 hours in the senior design project. The remaining 8 hours of the 64 hour core requirement cover some material of a more elementary nature, and reflect the overall slower pace appropriate to a less theoretically sophisticated clientele (as indicated by the low level of mathematical prerequisites).

The core materials (see figure 2) is organized along the 3 curriculum threads so that both maturity and integration will be achieved. During the freshman year two courses in programming (161-2), one in hardware (151), and one in systems (121) are specified. The programming sequence employs PL/C and emphasizes problem solving and professional programming methodology (7,8,11,12). At the end of the freshman year a second language is elected from the introductory language group according to professional interests. Although these languages are service oriented, they are all taught as second languages, and hence move rapidly through syntatical features and achieve significant exposure to the language's capabilities in one quarter's time.

The sophomore sequence prescribes a 3 course sequence in programming (261-2,361), which develops PL/C and PL/I to the point where a quarterlong small group project can be executed efficiently. Concurrently, the student takes the introductory systems software course (241) and the second systems course (221). By the end of the sophomore year the major knows a very powerful language very well, and has begun to appreciate the difficulties inherent in large programs and meaningful applications.

The junior sequence in the core (371-2,375) is aimed at developing multi-lingual capabilities and an understanding of the roles and interations of a variety of high-level and low-level languages. The major is also ready to appreciate the economic and management aspects of running a computer facility (320).

The core is completed in the senior year with a computer architecture course (421) and the senior software engineering project. This project is the synthesis of the lessons learned in applications programming and systems design. Projects aim at solving real problems and are usually industrially sponsored.

The packaging of the curriculum in quarter courses forced us to distribute topics over more courses, but it also allowed us to collect enough elementary material to offer three entry points into the curriculum through the service area. The overlap of major core courses and the service area (which does not include the major's programming sequence (13)) required a stratified curriculum rather than the spiral packaging illustrated by ACM'78. Central concepts are repeated and amplified as one rises through the curriculum, but technical concepts of concern only to the professional are minimized in the service area.

FIGURE 2. Computing Science Course Titles
(* Indicates A Core Course)

SYSTEMS

121* Introduction to Systems/4 hrs.
221* Information Systems/4 hrs.
320* Software Engineering and Computer Management /4 hrs.
421 Data Base Design and Implementation/4 hrs.
494-7* Software Engineering Project/1,2,3,1 hrs.

PROGRAMMING AND LANGUAGES

General Education
101 Survey of Computer Applications/2 hrs.
105 Algorithmic Processes/2 hrs.
106 Introduction to BASIC/2 hrs.

Introductory Languages
271 FORTRAN Programming/4 hrs.
272 COBOL Programming/4 hrs.
273 RPG II Programming/4 hrs.
274 PL/I programming/4 hrs.
275 S/370 Assembler Language Programming/ 4 hrs.
279 JCL and Utilities/4 hrs.

PL/I Sequence
161-2* Introduction to Programming/4,4 hrs.
261* Data Structures/4 hrs.
262* File Structures/4 hrs.
361* Advanced Programming Techniques/4 hrs.

Advanced Languages
371-2* Comparative Programming Languages/4,4 hrs.
375* Assembly Level Programming/4 hrs.
425 Modeling and Simulation/4 hrs.

HARDWARE/SOFTWARE
151* Introduction to Computer Hardware/4 hrs.
241* Introduction to Systems Software/4 hrs.
341 Compiler Design/4 hrs.
345 Operating Systems/4 hrs.
350 Digital Logic/4 hrs.
355 Introduction to Microcomputers/4 hrs.
450 Introduction to Data Communications/4 hrs.
451* Computer Architecture/4 hrs.

TOPICS AND SEMINARS
100* Computer Science Orientation Seminar/0 hrs.
499 Topics in Computer Science/1-4 hrs.

While the core unifies the computing science majors, the degree requirements heed the recommendation of both IEEE and ACM to provide experiences outside the discipline and outside the classroom. Majors pursue a related area comprising at least 40 hours in some other discipline, and then base their senior project in this area. The majority of students opt for a related area in business. In addition to the computing science core they will complete most of the School of Business core (32 quarter hours) and then take at least 16 more hours in computing science (including COBOL, RPGII and JCL) and 8 upper division hours in business. The mathematics requirement for this major is college algebra and finite mathematics, as well as the 3 course sequence in statistics and quantitative methods included in the business core.

A second computing science major involves a related area in Applied Mathematics in a specific discipline. Along with the computing science core and 8 additional C.S. hours, the student completes 27 quarter hours of calculus, differential equations, linear algebra and numerial analysis, and 36 quarter hours in a discipline such as Physics, Psyschology, or Biology. The premise is that the graduate will be conversant with the techniques of analysis and modeling which utilize the computer, as well as having significant knowledge in a discipline which employs such models. The senior project provides the opportunity to synthesize these experiences.

The third option is an individualized major defined by the student, his advisor in the department, and other faculty members according to the student's interests. Cross disciplinary specializations in engineering areas, political science or philosophy might be built upon the computing science core under this option. Also possible is the selection of courses which would best equip a student to pursue a particular graduate program in computer science or a related discipline.

A further benefit of this design is that various levels of emersion in the computing science curriculum can be identified short of a major. The service area offers a variety of courses with minimal prerequisites so that an emphasis or minor can be obtained by say, a business major, without competing with upper division majors or focusing on narrow, technical topics. The two year degree requires the freshman and sophomore core sequences and provides a solid introduction to programming. A major may, if necessary, leave the program at the end of any year and take with him a useful, balanced exposure to the discipline.

CONCLUSION

In summary, we found both the ACM '78 recommendation and the IEEE computer Society report invaluable in our curriculum design efforts both in terms of content suggestions and organizational models. Moreover, we had little difficulty meshing what we felt were the strengths of these recommendations within the constraints imposed by our particular environment--which speaks to the appropriate centrality of the concepts they both identify (6). Adopting these central concepts did not inhibit us from incorporating the additional emphases which meet the needs of our students. We feel the distinctive features which are exhibited in our curriculum: the integrated related area, the core language emphasis, and the senior design project are compatible with the curriculum reports.

As our program grows, the offerings can be extended naturally and consistantly due to the fundamental strength of the underlying curriculum structure. We have well defined interfaces, a clear prerequisite structure, and descernable levels of technical development. Evaluation of the program is therefore relatively easy and the places where adjustments must be made can be readily identified. Likewise, the competencies of our majors in the various areas of our program can be more easily monitored so that we have a greater degree of quality control.

We therefore offer this curriculum as an example of the usefulness of curriculum models and as one illustration of how a model can be implemented in spite of outward differences imposed by environment. We also recommend the process of design which has been illustrated as a useful model for productive results.

REFERENCES

1. Aiken, R.M. "Summary of Comments Following SIGCSE Panel Discussion on 'Computer Science Graduates--An Industry/University Gap'," SIGCSE Bulletin (ACM) 4, 3 (Oct 1972), 37.

2. Austing, R.H., and G.L. Engel, "A Computer Science Course Program For Small Colleges," COMM. ACM 16,3 (March 1973), 139-147.

3. Austing, R.H., et. al, "Curriculum Recommendations for the Undergraduate Program in Computer Science," SIGCSE Bulletin (ACM) 9,2 (June 1977), 1-16.

4. Couger, J.D. "Curriculum Recommendations for Undergraduate Programs in Information Science," COMM. ACM 16,12 (Dec. 1973),727-749.

5. Curriculum Committee on Computer Science "Curriculum '68, Recommendations for Academic Programs in Computer Science," COMM. ACM 11,13 (March 1969), 151-197.

6. Engle, Gerald L. "A Comparison of the ACM/C^3S and the IEEE/CS Model Curriculum Subcommittee Recommendations," COMPUTER 10,12 (Dec. 1977), 121-123.

7. Gries, David "What Should We Teach In An Introductory Programming Course?" SIGCSE Bulletin 6,1 (Feb. 1974), 81-89.

8. Holt, R.C. et. l, "SP/k: A System for Teaching Computer Programming," COMM. ACM 20,5 (May 1977), 301-309.

9. IEEE Computer Society Education Committee, Model Curricula Subcommittee, A Curriculum in Computer Science and Engineering, 1977.

10. Mitchell, William, Design of Mathematics Curricula For The Small College, University Microfilms 74-291-80.

11. Powers, Michael J., "Structuring an Applied Computer Science Program," College Curriculum in Computer Science, Engineering and Data Processing, IEEE, 1978.

12. Schneider, G. Michael, "The Introductory Programming Course in Computer Science--Ten Principles," SIGCSE Bulletin (ACM) 10,1 (Feb. 1978), 107-114.

13. Fisher, Hankley, Wallentine, "Seperation of Introductory Programming and Language Instruction," SIGCSE Bulletin (ACM) 5,1 (Feb. 1973), 9-13.