A LOOK BACK:
UNDERGRADUATE COMPUTER SCIENCE EDUCATION:
A NEW CURRICULUM PHILOSOPHY & OVERVIEW

John C. Knight, Jane C. Prey, & Wm. A. Wulf
Department of Computer Science
University of Virginia

ABSTRACT

As the discipline of computer science grew, undergraduate computer science education continued to change and, has for the most part, kept pace with the new topics in the field. However, the *pedagogy* had not changed significantly. Although the curriculum at the University of Virginia was comparable to that of most other universities, it had neither the rigor nor the practical experience needed to prepare undergraduates for the workplace or meaningful graduate study.

Thus, we believed a major shift of emphasis away from the traditional computer science curriculum was needed. We believe that

- a more engineering orientation must be incorporated into all of our core courses,
- students need a more extensive grounding in software engineering,
- all courses must increase the degree of mathematical rigor,
- students need hands-on experiences with appropriate, current artifacts, as well as
- intense laboratory experiences which will help students develop inter-personal and engineering skills in additional to course content, and
- providing real-world "practice" is important and needs to be a fundamental element of the complete curriculum.

We have incorporated these concepts into our new CS curriculum. We have offered the new CS1 course for 10 semesters with good results; we have also developed and offered three follow-on courses. The student and faculty responses have been very favorable. They are excited by the new courses and the closed laboratory component. All of these courses have lecture slides, laboratory activities, homework assignments, etc. available on the web for public viewing. A textbook for CS1 authored by the two UVA faculty most directly responsible for our CS1 course has recently become available.

INTRODUCTION

Computer science has changed rapidly in its brief history. What once were acceptable as skill and knowledge levels are now inadequate. Today's computer professional must have an extensive set of skills and detailed knowledge of many technical areas. Similarly, the individual entering the research community needs a thorough grounding in many diverse areas. The intent of our new curriculum is to prepare these professionals and researchers for their careers following graduation.

The Department of Computer Science at the University of Virginia is part of the School of Engineering and Applied Science. As such the Department has a strong commitment to achieving a true sense of rigorous engineering in our educational culture. We seek to educate computer scientists with a clear understanding of, an appreciation for, and skills that support the engineering and comprehension of large software systems, reengineering of existing systems, use of modern tools and environments, and application of innovative techniques such as software reuse.

To help realize these goals, we received a 3-year NSF curriculum development grant (CS-NSF-5239-92) starting July 1, 1992. This grant helps to support the development efforts for four of our core courses.

We have completed the major development work on the core courses of the new curriculum. The 1CS - Introduction to Computer Science was offered beginning the Fall, 1992 semester. The second course of the core (2SW - Software Development Methods) was offered beginning this Fall, 1993 semester. The third course (2PDR - Program and Data Representation) began in the Spring, 1994. The fourth course included in the grant (3SW - Advanced Software Development) was first offered in Spring, 1995.

As with any curriculum development, we do not believe we are really 'done.' We continue to modify current activities and develop new ones for the core courses while evaluating the application of the closed laboratory model for other required and elective courses. Presently we are discussing the development of closed laboratory type courses for our upper division courses, including networks, translation systems and CAD.

To gain a better understanding of what we are trying to do with our new curriculum, we developed this curriculum philosophy and a curriculum overview. We are pleased to be able to share this information with the attendees of the FIE '97 NSF Project Showcase.

CURRICULUM PHILOSOPHY

Where Are We?

As computer science grew as a discipline, undergraduate computer science education continued to change and, has for the most part, kept pace with the new topics in the field. However, the *pedagogy* has not changed significantly. At the University of Virginia, we have a curriculum that effects what is a common approach to teaching students. Curricula like ours emphasize:

- The construction of relatively small programs, at most a few hundred lines.
- The use of a programming language that is used rarely outside of undergraduate courses.
- The development of programs ‘‘afresh’’ for each assignment or course.
- A development environment lacking modern tools.
- Programming in isolation or in small groups at best.
- The belief that if a program ‘‘works’’ it is acceptable.
- An informal development approach rather than one that is rigorous and exercises analytic skills.

Why Change?

Comparing the content of the curriculum with the situation in the real world, we see a considerable contrast. Practicing computer scientists have to deal with the antithesis of what we teach:

- Software systems that are often thousands or even millions of source lines long.
- Tasks that involve modifying such systems rather than developing them.
- Existing systems that might be very old but remain important and have to be maintained.
- Tool-rich working environments.
- Development efforts that are undertaken by large teams.
- The realism of cost/performance trade-offs in business contexts.
- System development according to mandated processes and standards.
- Expenditure of considerable effort on tasks other than source-code development.

This list illustrates the general range of skills needed by a contemporary computing professional. This set of skills is the antithesis of what we are teaching. Clearly, there is a serious mismatch between what is taught, how it is taught, and the emphasis it receives on one hand, and what the consumers of the education actually need on the other.

Our new curriculum is driven by the desire to make the necessary changes to accommodate the educational needs of the future computing professional. These changes necessitate a complete revision of the content, approach, and resources used in the undergraduate teaching program.

What Is The Approach?

The Computer Science Department at the University of Virginia has undertaken a shift of emphasis in the computer science curriculum to meet the needs outlined above:

- A philosophy of engineering is incorporated into all of the core courses.
- An emphasis on software engineering begins in the very first course and continues.
- A high degree of mathematical rigor especially in discrete mathematics is included in the form of (a) new mathematics courses and (b) increased use of mathematics in other courses.
- A strong prerequisite structure emphasizes the interdependence of the material.
- Hands-on experience is facilitated by the development of a closed laboratory.
- Real-world artifacts are included to the extent possible in assignments and projects.
- Inter-personal and engineering skills are developed through laboratory and other projects.
- Emphasis is on use of knowledge and skills as they are acquired.

As well as having these characteristics, the approach includes systematic definitions of the contents of courses. These definitions take the form of detailed lists of topics that will be addressed by the course. The lists are separated into sublists of knowledge areas and skill areas, and the treatment of each topic is summarized as either resulting in ‘‘mastery of’’, ‘‘familiarity with’’, or ‘‘exposure to’’ the specific topic.

What Is The Impact Of These Changes?

The incorporation of these ideas into our teaching program has lead to the development of an entirely new undergraduate curriculum. The core courses of the new curriculum are designed to be more mathematically rigorous, more practice-oriented, more closely related to the real-world environment. This has lead to a number of substantial effects on our program, specifically:

- Extensive revision of many existing courses. All of the core courses containing elements of programming, data structures, machine representation, etc. have had to be revised extensively.
- Development of several new courses, especially in discrete mathematics and computation theory.
- The replacement of the programming language used for teaching purposes. C++ was selected as the new programming languages and, although not ideal, it has the benefits of supporting object-oriented programming and increasing industrial acceptance.
- A belief in the importance of reuse of courseware. We intend to make our artifacts available to other universities that choose to adopt elements of our curriculum.

- The development of the “closed laboratory” facility to support closed-laboratory exercises. The facility is being expanded to include devices such as motorized vehicles and other elements that simulate realistic applications.

CURRICULUM OVERVIEW

The philosophy of our new undergraduate computer science curriculum is a product of our entire faculty. We spent a significant amount of time discussing, disagreeing and deciding the best course for our students and faculty. Once our philosophy was established, a curriculum structure was necessary.

Among the more important decisions made relevant to our curriculum structure are:

- more rigor must be introduced at all levels of the curriculum,
- the early introduction of relevant mathematics and its continued use in subsequent courses,
- earlier introduction of software engineering concepts and their application in subsequent courses,
- a conscious effort to provide more out of classroom opportunities for undergraduates to interact with faculty and graduate students, and
- a focus on the practice of computing (by providing a closed laboratory course in the program of study every semester of the first three years.)

These decisions arose from a variety of considerations, but principal among them was our desire to teach modern computing methods using practical tools and applications from the first day of this new curriculum.

Curriculum Structure

The resulting structure of courses is shown in Figure 1. Figure 1 illustrates the relationship between the four courses for which new content and laboratory experiences are being developed under the NSF grant (noted with an asterisk) and the other required computer science courses in the curriculum. (Students are also required to take four additional CS courses from a set of elective upper division courses.) One of the more unique features of our program of study is the engineering school requirement of a senior thesis. The combining of the thesis with the senior seminar and the student’s opportunity to focus for the three previous years on the art of computing will result in what we believe to be a complete computer science experience.

Course Content Definition

After an extensive analysis of what we considered prerequisite knowledge and skills for our students to have before they graduated, we identified and prepared a detailed analysis of the required knowledge and skills which are to be learned in each particular course. Included in the analysis is the recognition that everything presented cannot be expected to be mastered, that it is appropriate to simply introduce concepts and skills which will be studied more in depth in later courses.

Our course content analysis resulted in a three-tier “learning frame.” We divided the knowledge and skills appropriate for the course into 1) mastery, 2) familiarity, and 3) exposure. Mastery implies a thorough understanding of the concept or skill and the ability to identify and select this alternative as the best choice in a particular situation without prompting from the instructor. Familiarity requires a good understanding of the concept or skill and the ability to apply it to a similar situation or when guided to do so by the instructor. Exposure is an introduction to the concept or skill, at least to the extent of recognizing its existence and utility but not necessarily its application.

Figure 2 is the course specification statement for core course 1CS. The remaining course specification statements (for 2SW, 2PDR, and 3SW as well as the other courses in our curriculum) can be obtained from the UVA CS department homepage (http://www.cs.virginia.edu). The course specification statements for 1CS, 2SW, 2PDR and 3SW are more detailed that the other courses in the curriculum since they have been under active development.

The curriculum committee is overseeing the process, but the entire faculty is engaged in the discussion and decision-making with our students offering significant insight. Many faculty members are devoting significant amounts of time to ensure that well-thought out, engaging and rigorous courses are available to our students.

STUDENT COMMENTS

While developing our core courses, we have asked and received many student comments. Students provided both formative and summative evaluation information for each course concerning every type of activity -- every laboratory exercise, lecture slides and content, etc.

The graduating class of May, 1996 was the first to complete all of the new core curriculum. One year after graduation, we asked them to reflect on their undergraduate experience. The following are a sample of the comments we received.

From those students who went into industry after graduation:

Q1. What parts of your UVA experience have been the most useful during your first year at work?

A1. Team work experience, having a good knowledge base on many CS topics (most other CS majors only had good OS or PL experience whereas our curriculum covered most everything), and a good grasp on the entire software development cycle using OO design and C++ application.

A2. I think that the group projects have been the most valuable to me in my work experience thus far. On my project, everything is a collaborative effort between designers, developers, and testers. All of the classes (especially 340 [3SW]) really helped to prepare me for this type of group environment and how everyone must work together to achieve the goal.

A3. ... The most essential stuff has been having a thorough background in OO programming, data structures, and algorithms. But I'm in a much more programming intensive and research oriented position than a typical CS job. In fact, I'm surprised to say I actually found myself breaking out my old probability and discrete math books (ie they were actually useful -- don't tell anyone I said this!) when I had to read a stack of graduate research papers in preparation for a project. ...

Q2. How would you compare your preparation for your job with others with CS degrees?

A1. Our program was definitely one of the best. In terms of classes offered and the amount of material we covered, there was no school that provided as much experience as the UVA program using C++ and Object Oriented Design.

A2. My first week on the job, I took a class with a recent XXX [major research university] CS grad who had graduated at the top of her class. During the class, I was much more involved in what was going on and how things worked than this other employee and was able to make an immedaite great impression on my new boss. ...

Q3. Other comments?

A1. Strongly encourage undergrads to take part in TAing. I personaly feel that the experience of working with and explaining concepts to other people has lead to gaining a reputation as being a problem solver that others can come to to help find a solution

Comments from students who enter graduate school in computer science.

Q1. What parts of your UVA experience have been the most useful to you in grad school?

A1. As always, knowing how to communicate (e.g., being able to a report about some technical concept or mechanism so someone can actually understand you). Being comfortable in front of a group when giving a presentation. Having been exposed to a good variety of CS research domains so that I know a little about everything (if nothing more than how to pronounce it).

Q2. How would you compare your preparation for grad school with others in your program with CS degrees?

A1. I think my preparation for grad school has been excellent. In general I feel a lot more confident in what I'm doing than I think most of my classmates are. ...

CONCLUSIONS

It is difficult to recognize when and where a curriculum is not meeting the needs of the student; it is even more difficult to recognize it and commit to change. The Computer Science faculty at the University of Virginia has recognized the weaknesses in its curriculum and has pledged to implement the changes needed. Our goal is to provide our students with a superior undergraduate education which will prepare them equally well for industry or graduate education.

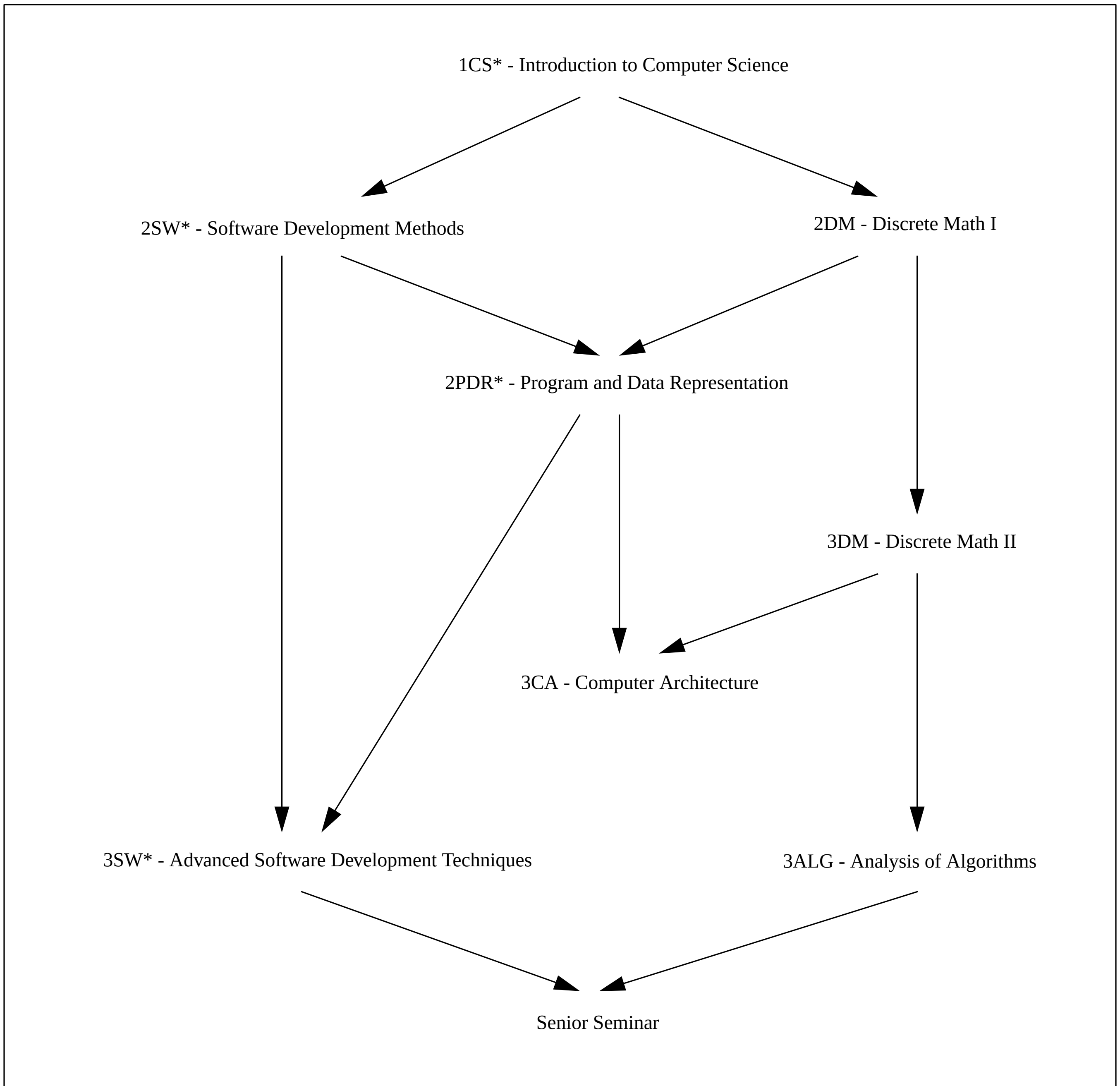
In the quickly changing computing environment, teaching computer science in the traditional manner does not prepare the student as effectively as we would like. We believe that a change in pedagogy provides the highest leverage for improving Computer Science education. With the revisions to our curriculum, students are learning modern computing methods using practical tools and applications from the beginning. Continuing this philosophy throughout the new curriculum, we will produce computer professionals who are knowledgeable not only in content, but who have practical experience in the workings of the real-world environment.

We are in various stages of development with our courses. Our first runs of the core courses in our new curriculum indicate we are on the right track. We intend to continue this dedication and commitment through the entire curriculum development and implementation process. We are presently continuing to revise these core courses as well as develop laboratory components for various upper division elective courses.

The current status of the curriculum can be found from our home page - http://www.cs.virginia.edu. A link to the undergraduate curriculum can be found there. The current versions of all of the courses will be on-line, including lecture slides, laboratory activities, FAQs, sample exam questions, etc.

We welcome your comments on these materials. Please send email to Jane Prey at prey@virginia.edu.

Figure 1: Required Computer Science Courses


1CS* - Introduction to Computer Science
2SW* - Software Development Methods
2DM - Discrete Math I
2PDR* - Program and Data Representation
3DM - Discrete Math II
3CA - Computer Architecture
3SW* - Advanced Software Development Techniques
3ALG - Analysis of Algorithms
Senior Seminar

* Courses Supported by NSF Grant CS-NSF-5239-92

Figure 2: 1CS Course Specification

Knowledge	Skills
Software Engineering Familiarity: Timing Debugging Software Reuse Standard Libraries Testing Exposure: Software Life Cycle Documentation Walk-Through Ethics	**Software Engineering** Familiarity: Develop test sets for simple programs Recognize importance of unambiguous problem and design specification Appreciate standard libraries
OS and Environment Familiarity: DOS Commands DOS File System Exposure: Windows TCP/IP	**OS and Environment** Mastery: Log onto computer system Copy files Traverse directory system Familiarity: Edit files Manipulate windows Send electronic mail. Exposure: Retrieve files from other sites
Imperative Programming Mastery: Basic Data Types Variables Assignment Familiarity: Operator Precedence Iteration Conditional Statements Streams Reference and Value Parameters Structures Classes Exposure: Pointers	**Imperative Programming** Mastery: Pass arguments using the appropriate parameter style. Use appropriate control structures within a program. Familiarity: Implement a two-hundred or so line program given a problem specification and a detailed design Use a debugger to trace a program Read a 500 line C++ program Exposure: Time individual segments of a program.
Object-Oriented Programming Familiarity: Abstract Data Types Overloading Exposure: Polymorphism Templates Inheritance	**Object-Oriented Programming** Familiarity: Recognize importance of paradigm Use existing classes Design an abstract data type with appropriate information hiding Add function to an existing class.
Problem Solving and Analysis Exposure: Top-Down Design Computational complexity	
Data Representation Mastery: Arrays Strings Familiarity: Structures Objects File Processing	