

# 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  
Charlottesville, VA 22903

## 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 our first course of the new curriculum (1CS: Introduction to Computer Science) and offered it beginning the Fall, 1992 semester. The second course of the core (2SW: Software Development Methods) has just been completed and will be offered beginning this Fall, 1993 semester. The third course, 2PDR: Program and Data Representation is presently being developed and will be offered starting Spring, 1994. The development of the fourth course included in the grant, 3SW: Advanced Software Development will begin during Summer, 1994 and offered Fall, 1994.

To gain a better understanding of what we are trying to do with our new curriculum, we developed a curriculum philosophy and a curriculum overview. We have shared

these documents with our NSF Visiting Committee at our annual review meeting this past March. We would like to share much of the same information with the ACMCSE members.

## 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 pro-

cesses 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 led 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 led 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 this grant (noted with an asterisk) and the other computer science courses in the curriculum. 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) can be obtained from the authors. The course specification statements for 1CS, 2SW and 2PDR are more detailed than the other courses in the curriculum since they have been under active development. A similar definition development is scheduled to begin Spring, 1994 for 3SW with implementation Spring, 1995. The remaining courses (none of which is included in the grant) will be scheduled for development and implementation beginning in the next three years.

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

## **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 first two 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.

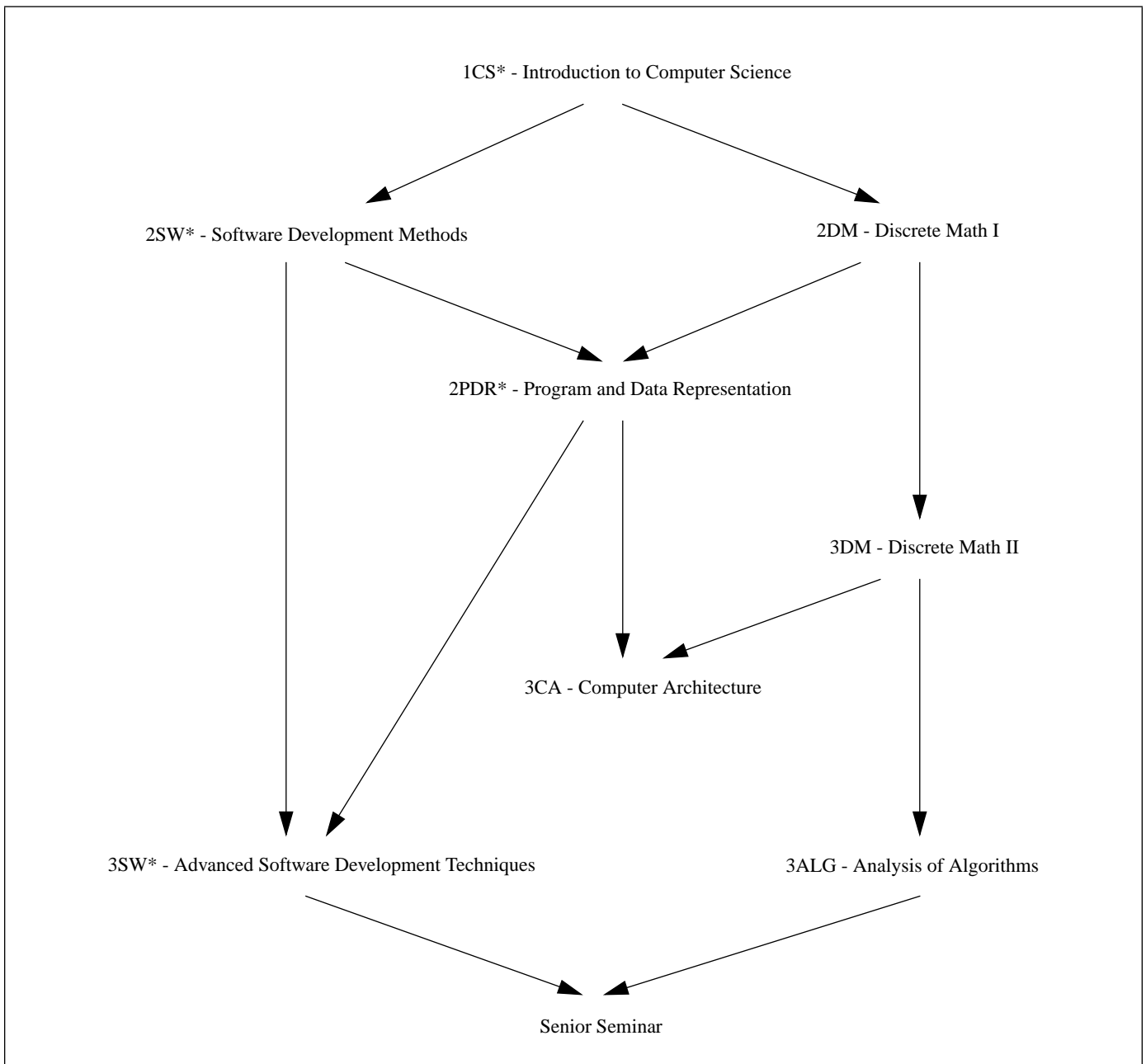


Figure 1: Abridged Computer Science Course Structure

Knowledge	Skills
<b>Software Engineering</b>	
Familiarity	Timing Debugging Software Reuse Standard Libraries Testing
Exposure	Software Life Cycle Documentation Walk-Through Ethics
<b>OS and Environment</b>	
Familiarity	DOS Commands DOS File System
Exposure	Windows TCP/IP
<b>Imperative Programming</b>	
Mastery	Basic Data Types Variables Assignment
Familiarity	Operator Precedence Iteration Conditional Statements Streams Reference and Value Parameters Structures Classes
Exposure	Pointers
<b>Object-Oriented Programming</b>	
Familiarity	Abstract Data Types Overloading
Exposure	Polymorphism Templates Inheritance
<b>Problem Solving and Analysis</b>	
Exposure	Top-Down Design Computational complexity
<b>Data Representation</b>	
Mastery	Arrays Strings
Familiarity	Structures Objects File Processing
<b>Software Engineering</b>	
Familiarity	Develop test sets for simple programs Recognize importance of unambiguous problem and design specification Appreciate standard libraries
<b>OS and Environment</b>	
Mastery	Log onto computer system Copy files Traverse directory system
Familiarity	Edit files Manipulate windows Send electronic mail.
Exposure	Retrieve files from other sites
<b>Imperative Programming</b>	
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.
<b>Object-Oriented Programming</b>	
Familiarity	Recognize importance of paradigm Use existing classes Design an abstract data type with appropriate information hiding Add function to an existing class.

Figure 2: 1CS Course Content.