Computing Curriculum 2001: First-Year Options for Computer Science majors

Thomas P. Wiggen
Computer Science Department
University of North Dakota
Grand Forks, ND 58202-9015
wiggen@cs.und.edu

Abstract

This paper compares the two most popular "programming-first" options for the beginning course sequence for Computer Science majors. The course descriptions and bodies of content for these options are thoroughly described in the final report of the Computing Curricula 2001 project, a joint undertaking of the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM).

The objects-first and imperative-first paradigms for the beginning programming sequence for CS majors are compared, with the goal of helping computer science curriculum planners to choose the best option for their own institution and students. No attempt is made to determine which of these options is generally better; in fact, that determination is most likely not possible. The focus is, rather, on the similarities, differences, and trade-offs between these approaches. Most of our curriculum design effort is expended on the design of new upper-level courses for majors, so the beginning sequence gets less attention as a result. The support (via textbooks, programming languages, code libraries and development environments) for each of these paradigms is also an important consideration when a faculty needs to choose amongst them.

There are two sub-options within these two programming-first approaches: the 2-semester sequence and the 3-semester sequence. This is an increasingly important consideration for many computer science departments because the desire to include the design and use of graphical user interfaces as a part of these courses has the side-effect of reducing the amount of time available for some of the other "traditional" elements of this sequence such as recursion and data structures.

The remaining, less commonly adopted, options described in CC2001 for the first-year sequence are breadth-first, functional-first, hardware-first and algorithms-first and they won't be included in this comparison.

1. Introduction

We could argue that the first year of coursework for majors in Computer Science is the most crucial for our students. This is the course that is supposed to *get them off the ground*. This is where our students will acquire skills and learn concepts that will enable them to survive, if not flourish, in their intermediate level Computer Science courses. Although we've had standardized descriptions of CS1 and CS2 content for a number of years, at times, we seem to be puzzled about what we should be doing in these courses. Perhaps this is because the rapid changes in the hardware, software and languages that underlie the *skills portion* of our curriculum.

The final report of the Computing Curricula 2001 project [1] (hereafter referred to as CC2001), a joint undertaking of the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM) identifies a number of different models for beginning and intermediate coursework in Computer Science.

The introductory coursework described in CC2001 includes both 2-semester and 3-semester sequences, with different emphases labeled as imperative-first, objects-first, functional-first, breadth-first, algorithms-first and hardware-first. The intermediate-level coursework can follow a traditional topic-based approach, a compressed approach, a systems-based approach and a novel web-based approach. The CC2001 report includes course descriptions and bodies of content for these options.

The CC2001 task force lists eleven principles that guided their work. They are:

1. Computing is a broad field that extends well beyond the boundaries of computer science.
2. Computer science draws its foundations from a wide variety of disciplines.
3. The rapid evolution of computer science requires an ongoing review of the corresponding curriculum.
4. Development of a computer science curriculum must be sensitive to changes in technology, new developments in pedagogy, and the importance of lifelong learning.
5. CC2001 must go beyond knowledge units to offer significant guidance in terms of individual course design.
6. CC2001 should seek to identify the fundamental skills and knowledge that all computing students must possess.
7. The required body of knowledge must be made as small as possible.
8. CC2001 must strive to be international in scope.
9. The development of CC2001 must be broadly based.
10. CC2001 must include professional practice as an integral component of the undergraduate curriculum.
11. CC2001 must include discussions of strategies and tactics for implementation along with high-level recommendations.

Of particular note is principle #5: *CC2001 must … offer significant guidance in terms of individual course design* which the task force report elaborates on as follows: "*Although the knowledge-unit structure used in CC1991 can serve as a useful framework, most institutions need more detailed guidance. For such institutions, CC2001 will be effective only to the extent that it defines a small set of alternative models—preferably between two and four—that assemble the knowledge units into reasonable, easily implemented courses. Articulating a set of well-defined models will make it easier for institutions to share pedagogical strategies and tools. It will also provide a framework for publishers who provide the textbooks and other materials for those courses*."

Thus, one of CC2001's goals, as stated above, is to provide us with a small set of alternatives. For the introductory sequence, the major choice appears to be between objects-first and imperative-first approaches. Our focus for the remainder of this paper will be on the comparison of these two alternatives and on the length of introductory sequence.

2. The Length of the Introductory Computing Sequence

There was a time when Calculus, the introduction to the mathematics major, was a 2-semester sequence. It was preceded, for most students, by a course in Analytical Geometry. The now-standard 3-semester Calculus sequence came about when Analytical Geometry was absorbed into the Calculus sequence.

There was time when University Physics, the introduction to the physics major, was a 2-semester sequence. It was followed, for most students, by an elective course in Modern Physics (relativity and quantum physics). At many universities, the now-standard 3-semester introductory physics sequence for majors came about when Modern Physics was absorbed into the introductory sequence.

Has something similar happened, or is something similar happening, to Computer Science. Is it time for our introductory sequence, typically with a 2-semester duration, to swallow up an additional chunk of content and evolve into a 3-semester sequence. The CC2001 task force strongly believes that this is the case for most institutions, though it is hard to describe exactly what that "additional chunk of content" is. Rather than being one big thing, it is more likely to be described as a lot of little things. According to the CC2001 task force, "*the number and complexity of topics that entering students must understand have increased substantially, just as the problems we ask them to solve and the tools they must use have become more sophisticated*."

As a result, the old concept of the CS1-CS2 sequence is being replaced by the new concepts of the 3-semester CS101-CS102-CS103 sequence and an alternative 2-semester CS111-CS112. We should all learn to use these new sequence numbers in our future discussions about the introductory computer science sequence. It's time for all of us, if we haven't already done so, to examine our introductory sequence and see which model is better for our curriculum. Would a 3-semester introductory computing "sequence"

mean three courses from three (or more) different textbooks? Probably so! Thus, it is an awkward kind of sequence for many of us, and perhaps to many students as well. The word "sequence" implies some sort of continuity. In reality, we often seem to have a jump discontinuity between CS1 and CS2 (excuse me, I meant to say "between CS111 and CS112") as we switch from the first-semester author/text to the second-semester author/text. We can expect to have two jump discontinuities in the 3-semester chain (one between CS101 and CS102, the other between CS102 and CS103).

3. Objects-First vs. Imperative-First

An algorithms-first approach would teach programming via pseudocode rather than by choosing a "real" programming language. This reduces the syntax burden on students, but also gives them no easy way to verify the correctness of their algorithms. Thus, the use of an executable programming language is the only realistic choice for most of us. Today, that usually means Pascal, C++, Java, C# or C (though I'm not really sure anyone really uses Pascal any longer … and whatever happened to Ada and Modula-2?). There's no getting around the fact that our Computer Science majors need to learn Object-Oriented programming. This is the dominant paradigm of our time, and cannot be ignored or omitted. Thus, an object-oriented language should be used regardless of the approach chosen. Most of us will choose similar languages and use the programming-first approach. How will we proceed?

We've had discussions like these before. Recall, in the 1980's, the "subroutines-first" vs. "loops-first" conflicts of those days and the different genre of Pascal textbooks that sprang up around those two notions. It's like the difference between a top-down and a bottom-up approach to learning about programs. The subroutines-first idea was to show you the general form of a program (logic distributed across a number of modules) before you zoomed in to see the details of those modules. The loops-first idea was to show you all the nuts and bolts first, then discover how you could use them to built complex gadgets. The objects-first vs. imperative-first battle is just a continuation of that earlier struggle.

Faculty members I've known have usually had a strong preference for one of these approaches over the other, and have taught best when using their favorite approach. This suggests that the way to solve the problem is to reach a consensus with current faculty (or at least, with those who might get that teaching assignment) and establish a *departmental approach* which will be difficult, but not impossible, to change. It certainly doesn't seem right to switch between objects-first and imperative-first approaches each time a teaching assignment changes. This is a sequence of courses, and students will from time to time, drop out, restart, or re-enter the sequence.

The biggest differences between the imperative-first and objects-first 2-semester sequences are highlighted (bold face) in the table below, which consists of reformatted information taken from the CC2001 report. As you can see, the imperative-first approach spends more time on data structure implementations (graphs and trees), architecture, and

compilers. The objects-first sequence replaces these topics with human-computer interaction, event-driven programming, algorithmic strategies, and a risks & liabilities topic. The rest of the coverage is more-or-less the same, though the arrangement of those common units differs.

	Imperative-first: Units Covered	Objects-first: Units Covered
CS 111	PF1 Fund. programming constructs 9 hrs PF2 Alg. and problem-solving 3 hrs (of 6) PF3 Fundamental data structures 6 hrs (of 14) AL3 Fund. computing algorithms 2 hrs (of 12) AL5 Basic computability 1 hr (of 6) **AR2 Machine level repr. of data 1 hr (of 3)** **AR3 Assembly level machine org. 2 hrs (of 9)** PL1 Overview of programming lang. 1 hr (of 2) PL4 Declarations and types 1 hr (of 3) PL5 Abstraction mechanisms 2 hrs (of 3) PL6 Object-oriented programming 3 hrs (of 10) GV1 Fundamental techniques in graphics 2 hrs SP1 History of computing 1 hr SE1 Software design 2 hrs (of 8) SE3 Software tools and environments 1 hr (of 3) SE5 Software req. and spec. 1 hr (of 4) SE6 Software validation 1 hr (of 3) Elective topics 1 hr	PF1 Fund. programming constructs 7 hrs (of 9) PF2 Alg. and problem-solving 2 hrs (of 6) PF3 Fundamental data structures 3 hrs (of 14) PF4 Recursion 2 hrs (of 5) AL3 Fund. computing algorithms 3 hrs (of 12) AL5 Basic computability 1 hr (of 6) PL4 Declarations and types 2 hrs (of 3) PL5 Abstraction mechanisms 1 hr (of 3) PL6 Object-oriented programming 8 hrs (of 10) GV1 Fundamental techniques in graphics 2 hrs SP1 History of computing 1 hr **SP5 Risks & liabilities of comp. sys. 1 hr (of 2)** SE1 Software design 2 hrs (of 8) SE2 Using APIs 1 hr (of 5) SE3 Software tools and environments 2 hrs (of 3) Elective topics 2 hrs
CS 112	**DS5 Graphs and trees 2 hrs (of 4)** PF3 Fundamental data structures 6 hrs (of 14) PF4 Recursion 5 hrs AL1 Basic algorithmic analysis 2 hrs (of 4) AL3 Fund. computing algorithms 4 hrs (of 12) PL1 Overview of programming lang. 1 hr (of 2) PL2 Virtual machines 1 hr **PL3 Introduction to language translation 2 hrs** PL4 Declarations and types 2 hrs (of 3) PL5 Abstraction mechanisms 1 hr (of 3) PL6 Object-oriented programming 7 hrs (of 10) SE1 Software design 2 hrs (of 8) SE2 Using APIs 2 hrs (of 5) SE3 Software tools and environments 2 hrs (of 3) Elective topics 1 hr	PF1 Fund. programming constructs 2 hrs (of 9) PF2 Alg. and problem-solving 2 hrs (of 6) PF3 Fundamental data structures 8 hrs (of 14) PF4 Recursion 3 hrs (of 5) **PF5 Event-driven programming 2 hrs (of 4)** AL1 Basic algorithmic analysis 2 hrs (of 4) **AL2 Algorithmic strategies 2 hrs (of 6)** AL3 Fund. computing algorithms 3 hrs (of 12) PL1 Overview of programming lang. 2 hrs PL2 Virtual machines 1 hr PL4 Declarations and types 1 hr (of 3) PL5 Abstraction mechanisms 2 hrs (of 3) PL6 Object-oriented programming 4 hrs (of 10) **HC1 Human-computer interaction 1 hr (of 6)** SE1 Software design 2 hrs (of 8) SE2 Using APIs 1 hr (of 5) SE5 Software req.s and spec. 1 hr (of 4) SE6 Software validation 1 hr (of 3)

Figure 1: Comparison of 2-semester sequences

You can begin to see where a 3-semester sequence can give you time to even out the differences between these two approaches as well as time to cover additional topics that would promote the student's breadth of knowledge about computer science. Each of

these two semester sequences is still a programming-language dominated course with little time for non-programming topics.
A side-by-side comparison of the imperative-first and objects-first 3-semester 101-102-013 sequences results in the following table:

	Imperative-first: Units Covered	**Objects-first: Units Covered**
CS 101	PF1 Fund. prog. constructs 10 hrs (9 + 1) PF2 Alg. and problem-solving 3 hrs (of 6) PF3 Fund. data structures 2 hrs (of 14) AR2 Machine level repr. of data 1 hr (of 3) **AR3 Assembly level machine org. 2 hrs (of 9)** **OS1 Overview of operating systems 1 hr (of 2)** **NC1 Intro. to net-centric computing 1 hr (of 2)** **PL3 Intro. to Lang. translation 1 hr (of 2)** PL4 Declarations and types 3 hrs PL5 Abstraction mechanisms 3 hrs HC1 Found. of HCI 1 hr (of 6) GV1 Fund. techniques in graphics 1 hr (of 2) SP1 History of computing 1 hr **SP2 Social context of computing 1 hr (of 3)** SP4 Prof. and ethical responsibilities 1 hr (of 3) **SP6 Intellectual property 1 hr (of 3)** SE1 Software design 3 hrs (of 8) SE3 Software tools and env. 2 hrs (of 3) **SE4 Software processes 1 hr (of 2)** Elective topics 1hr	PF1 Fund. prog. constructs 9 hrs PF2 Alg. and problem-solving 3 hrs (of 6) PF3 Fund. data structures 3 hrs (of 14) AL3 Fund. computing Alg. 1 hour (of 12) AL5 Basic computability 1 hour (of 6) AR2 Machine level repr. of data 2 hrs (of 3) PL1 Overview of prog. Lang. 2 hrs PL4 Declarations and types 2 hrs (of 3) PL6 Object-oriented prog. 7 hrs (of 10) **PL8 Lang. translation systems 1hr** SP1 History of computing 1 hr SP4 Prof. and ethical responsibilities 1 hr (of 3) **SP5 Risks & liabilities of comp. sys. 1 hr (of 2)** SE3 Software tools and environments 1 hr (of 3) SE6 Software validation 1 hr (of 3) Elective topics 4 hrs
CS 102	PF1 Fund. prog. constructs 3 hrs (of 9) PF2 Alg. and problem-solving 6 hrs PF3 Fund. data structures 5 hrs (of 14) PF5 Event-driven prog. 1 hr (of 4) AL3 Fund. computing Alg. 3 hrs (of 12) AR2 Machine level repr. of data 2 hrs (of 3) PL1 Overview of prog. Lang. 1 hr (of 2) PL2 Virtual machines 1 hr **PL3 Introduction to Lang. translation 1 hr (of 2)** PL6 Object-oriented prog. 6 hrs (of 10) HC1 Found. of HCI 1 hr (of 6) **HC2 Building a simple GUI 2 hrs** **IM2 Database systems 1 hr (of 3)** SE1 Software design 1 hr (of 8) SE2 Using APIs 2 hrs (of 5) SE5 Software req. and specifications 1 hr (of 4) SE6 Software validation 1 hr (of 3) SE7 Software evolution 1 hr (of 3) Elective topics 1hr	PF3 Fund. data structures 3 hrs (of 14) PF4 Recursion 2 hrs (of 5) PF5 Event-driven prog. 2 hrs (of 4) SE2 Using APIs 2 hrs (of 5) AL1 Basic algorithmic analysis 1 hr (of 4) AL3 Fund. computing Alg. 2 hrs (of 12) AR2 Machine level repr. of data 1 hr (of 3) PL2 Virtual machines 1 hr PL4 Declarations and types 1 hr (of 3) PL5 Abstraction mechanisms 3 hrs PL6 Object-oriented prog. 7 hrs (of 10) HC1 Found. of HCI 1 hr (of 6) GV1 Fund. techniques in graphics 2 hrs SE1 Software design 3 hrs (of 8) SE3 Software tools and environments 1 hr (of 3) SE5 Software req. and specifications 1 hr (of 4) SE6 Software validation 1 hr (of 3) SE7 Software evolution 1 hr (of 3) Elective topics 5 hrs
CS 103	**DS5 Graphs and trees 2 hrs (of 4)** PF3 Fund. data structures 12 hrs (of 14) PF4 Recursion 5 hrs AL1 Basic algorithmic analysis 2 hrs (of 4) AL2 Algorithmic strategies 3 hrs (of 6) AL3 Fund. computing Alg. 5 hrs (of 12) AL5 Basic computability 1 hr (of 6) PL1 Overview of prog. Lang. 1 hr (of 2) PL6 Object-oriented prog. 8 hrs (of 10) SE6 Software validation 1 hr (of 3)	PF2 Alg. and problem-solving 3 hrs (of 6) PF3 Fund. data structures 11 hrs (of 14) PF4 Recursion 6 hrs (5 + 1) AL1 Basic algorithmic analysis 3 hrs (of 4) AL2 Algorithmic strategies 6 hrs AL3 Fund. computing Alg. 5 hrs (of 12) SE1 Software design 1 hr (of 8) **SE8 Software project management 1 hr (of 3)** Elective topics 4 hrs

Figure 2: Comparison of 3-semester sequences

There are a bunch of small differences and a few bigger differences in the recommended content of these 3-semester sequences. The bold-faced items in the tables above indicate topics with are listed in one sequence and not in the other. There are 11 bold-faced items in the column representing the imperative-first option and 3 under the object-first option. For the most part, these are minor 1-hour items that might be picked up as elective topics in the other column. The one that stands out as the biggest difference is DS5 Graphs and Trees, a fundamental topic in a typical introductory sequence, which receives 2 hours in the imperative CS103 but is not mentioned in the object CS103. Since the objects-first sequence allows more elective hours (13, compared with just 2 for the imperative sequence), this would surely be among those electives.

Some topics are treated for comparable, but not identical, lengths of time under both options. Among these are GV1 Fundamentals of Graphics (1 hour in the imperative option, 2 hours in the objects option), AL1 Basic Algorithm Analysis (2 hours in the imperative option, 3 hours in the objects option), and AL2 Algorithmic Strategies (3 hours in the imperative option, 6 hours in the objects option).

The programming fundamentals (PF) group of topics breaks down quite differently in the two options. A total of 47 class hours are dedicated to programming fundamentals under the imperative-first option, which 42 class hours are dedicated to these topics under the objects-first option. Moreover, the distribution of those hours amongst the five PF categories is somewhat differently shaped. Some time spent on programming constructs and algorithms under the imperative-first option is shifted to recursion and event-driven programming under the objects-first option. Here is a brief summary of the differences.

	Imperative: CS101+CS102+CS103=total	Object: CS101+CS102+CS103=total
PF1	10 + 3 + 0 = 13	9 + 0 + 0 = 9
PF2	3 + 6 + 0 = 9	3 + 0 + 3 = 6
PF3	2 + 5 +12 = 19	3 + 3 + 11 = 17
PF4	0 + 0 + 5 = 5	0 + 2 + 6 = 8
PF5	0 + 1 + 0 = 1	0 + 2 + 0 = 2

4. A Broader View: The Computing Disciplines

The CC2001 report on Computer Science curriculum [1] is now accompanied by a number of other similar reports for similar majors. IS2002 is the corresponding document for Information System curricula. SE2004 is the corresponding documents for Software Engineering curricula. CE2004 is the Computer Engineering curriculum report and IT2005 (in progress) will describe curricula for Information Technology programs. In fact, CC2001 now has the alias name of CS2001 to emphasis that it is specifically aimed as Computer Science, rather than at all the computing curricula. The designation CC20xx will henceforth be used to designate overview documents that take a broader view that encompasses all of these specific disciplines (and, perhaps, other computing

disciplines which are not known today). The first of these reports is CC2004 which is currently available in draft form [2].

In addition to having a number of viable choices for the design of the entry level sequence for computer science majors, we are also faced with the likelihood that that same entry level sequence will be used to introduce a majors in similar, but different degree programs, to computing. An unpleasant alternative is to have distinct entry level sequences for Computer Science (CS), Computer Engineering (CE), Information Systems (IS), Information Technology (IT) and Software Engineering (SE) majors (should all these programs exist on one campus).

The following table from [2] uses a scale from 0 to 5 to rank the relative importance of each of a number of topics for majors in these programs. Each topic got a minimum and maximum importance ranking, representing the consensus opinions of the members of the CC2004 Joint Task Force. For some knowledge areas covered by the introductory sequence, the ranking ranges barely overlap, or don't overlap at all. It is probably unrealistic to hope for one introductory sequence that can be shared by all of these computing disciplines.

Knowledge Area	*CE*		*CS*		*IS*		*IT*		*SE*	
	min	*max*	*min*	*max*	*min*	*max*	*min*	*max*	*min*	*max*
Programming Fundamentals	4	4	4	5	2	4	2	4	5	5
Integrative Programming	0	2	1	3	2	4	3	5	1	3
Algorithms and Complexity	2	4	4	5	1	2	1	2	4	4
Computer Architecture and Organization	5	5	2	4	1	2	1	2	2	4
Operating Systems Principles & Design	2	4	3	5	1	1	1	2	3	4
Operating Systems Configuration & Use	2	3	2	4	2	3	3	5	2	4
Net Centric Principles and Design	1	3	2	4	1	3	3	4	2	4
Net Centric Use and configuration	1	2	2	3	2	4	4	5	2	3
Platform technologies	0	1	0	2	1	3	2	4	0	3
Theory of Programming Languages	1	2	3	5	0	1	0	1	2	4
Human-Computer Interaction	2	5	2	4	2	5	4	5	3	5
Graphics and Visualization	1	3	1	5	1	1	0	1	1	3
Intelligent Systems (AI)	1	3	2	5	1	1	0	0	0	0
Information Management (DB) Theory	1	3	2	5	1	3	1	1	2	5
Information Management (DB) Practice	1	2	1	4	4	5	3	4	1	4
Scientific computing (Numerical mthds)	0	2	0	5	0	0	0	0	0	0
Legal / Professional / Ethics / Society	2	5	2	4	2	5	2	4	2	5
Information Systems Development	0	2	0	2	5	5	1	3	2	4
Analysis of Technical Requirements	2	5	2	4	2	4	3	5	3	5
Engineering Foundations for SW	1	2	1	2	1	1	0	0	2	4
Engineering Economics for SW	1	1	1	1	2	2	1	1	2	2
Software Modeling and Analysis	1	3	2	3	3	3	1	3	4	5
Software Design	2	4	3	5	1	3	1	2	5	5
Software Verification and Validation	1	3	1	2	1	2	1	2	4	5
Software Evolution (maintenance)	1	3	1	1	1	2	1	2	2	4
Software Process	1	1	1	1	1	1	1	1	2	4
Software Quality	2	3	2	3	1	2	1	2	3	4
Comp Systems Engineering	5	5	1	2	0	0	0	0	2	3
Digital logic	5	5	2	3	1	1	1	1	0	3
Distributed Systems	3	5	1	3	2	4	1	3	2	4
Security: issues and principles	2	3	1	4	2	3	1	3	1	3
Security: implementation and mgt	1	2	1	3	1	3	3	5	1	3
Systems administration	1	2	1	1	1	3	3	5	1	2
Systems integration	1	4	1	2	1	4	4	5	1	3
Digital media development	0	2	0	1	1	2	3	5	0	1
Technical support	0	1	0	1	1	3	5	5	0	1

Figure 3: Comparative weights of computing topics across degree programs

5. Practical Considerations

We are all encouraged by the CC2001 task force to re-examine our introductory sequences in Computer Science. They expect that the 3-semester sequence will become increasingly common and they are encouraging departments and individual faculty to experiment with these models and other similar models of their own design.

Today, it seems that most new Java books are written for the objects-first option. You can judge this by their focus on the use of the Java API and Java Collections objects rather than on the implementation of these data structures by the student. Most Java texts will include complete implementation code for stacks, queues and lists, leaving no meaningful homework assignments for students along these lines. A typical imperative-first sequence would, at some point in time, assign students the task of writing their own implementations of a stack and a queue. A typical objects-first sequence would, at some point in time, assign students the task of using a stack or queue object (from the Java API, for example) to accomplish some other purpose. Object-first students would be taught to use stacks and queues just as they use integers and floats (i.e., as built-in stuff). In a certain sense, this seems to be right. It is better to stand on the shoulders of others than to re-invent a lot of wheels. On the other hand, many Computer Science faculty remember how educational it was to learn (gradually) how to encapsulate and hide information. It would be a shame (wouldn't it?) if tomorrow's students couldn't feel the thrills that we felt when we first used the "compile only" option.

Which way should we go? Two semesters or three semesters? Objects first or objects later? It looks like the jury is still out on these questions. We all need to take a close look at what we are doing now and determine the path that is better for the futures of our programs and our students. It won't be easy, but it is important and it might be fun.

References

[1] Computing Curricula 2001: Computer Science, Final Report, by the ACM/IEEE-CS Joint Task Force on Computing Curricula, December 2001.

[2] November 22, 2004 Draft of the Computing Curricula 2004 Overview Report, by the ACM/AIS/IEEE-CS Joint Task Force for Computing Curricula 2004.