A Model Curriculum for K-12 Computer Science:
Report of the ACM K-12 Education Task Force Computer Science Curriculum Committee
Draft – not for distribution or citation – 11/1/02

Allen Tucker (editor) – Bowdoin College
Fadi Deek – New Jersey Institute of Technology
Jill Jones – Carl Hayden High School
Dennis McCowan – Weston Public Schools
Chris Stephenson – University of Waterloo
Anita Verno – Bergen Community College

Contents **Page**

1. Introduction

The purpose of this report is to define a model curriculum for K-12 computer science and to suggest steps that will be needed to enable its wide implementation. The goal of such a curriculum is to introduce the principles and methodologies of computer science to all students, whether they are college- or workplace- bound.

Much evidence [NAS99] confirms an urgent need to improve the level of public understanding of computer science as an academic and professional field, including its distinctions from information technology (IT), mathematics, and the other sciences. Elementary and secondary schools have a unique opportunity and responsibility to address this need. A broad commitment to K-12 computer science education will not only improve public understanding but also help to address the worldwide shortage of computer specialists. The creation of a viable model for teaching computer science at the K-12 level is a necessary first step toward reaching these goals.

This report's recommendations address the entire K-12 range, rather than limiting itself to grades 9-12. Moreover, it complements existing K-12 computer science and IT curricula where they are already established, especially the advanced placement (AP) computer science curriculum [APCS02] and the national educational technology standard (NETS) curriculum [NETS02].

At this time, the development of state-level curriculum standards for computer science in the US is nearly nonexistent. Some state standards now identify "information technology" as a subject area - either stand-alone (e.g., Arizona's use of the NETS curriculum) or as a collection of topics integrated with other science curricula (e.g., Maine's "Learning Results" [LR]). An important goal of this report will be to provide all states with a comprehensive framework that can be used for incorporating computer science into their existing curriculum standards.

Future drafts of this report will be informed by feedback from many sources, and its success depends upon widespread dissemination and constructive criticism from persons who have interests or experience in K-12 computer science education. To this end, a Web site has been established (see http://www.acm.org/k12) for disseminating drafts of this report and gathering feedback from the community. This report will be finalized during the summer of 2003.

Feedback will be actively sought from several professional groups, including the following:

ACM Education Board
ACM SIGCSE
ISTE Special Interest Group for Computer Science (SIGCS)
Directors of curriculum in school districts (ASCD)
National Education Association (NEA)
National Association of Secondary School Principals (NASSP)
National School Board Association
Academy of Information Technology/National Academy Foundation (AOIT/NAT)

In addition, presentations at the NECC, FIE, and SIGCSE Symposia will be used for feedback and dissemination. Publication of our final draft in journals like CS Education, SIGCSE Journal, the Journal of Computer Science Education, and CACM, as well as on the Web, will be the final outcome of this committee's work.

2. Background

As a basis for describing a model curriculum for K-12 computer science, we use the following definition of computer science as an academic and professional field.

> *Computer science* (*CS*) is the study of computers and computational processes (known as "algorithms"), including their principles, their hardware and software designs, their applications, and their impact on society.

> An *algorithm* is a precise description of a solution to a computational problem. *Programming* is used to implement algorithms. [1]

In addition to programming, computer science includes the study of hardware, networks, graphics, databases and information retrieval, computer security, software engineering, programming languages, programming paradigms, translation between levels of abstraction, artificial intelligence, the limits of computation (what computers *can't* do), applications, and social issues (Internet security, privacy, intellectual property, etc.).

Typically, K-12 science and mathematics curricula do not cover these topics. However, some of the emerging K-12 information technology (IT) curricula are addressing some of them, especially the applications and social impact of computers. However, there is strong evidence [NAS99] that a basic understanding of all these topics is now an essential ingredient to preparing high school graduates for life in the 21st century.

The goals of a K-12 computer science curriculum are threefold:

1) to introduce the fundamental concepts of computer science to all students, beginning at the elementary school level.
2) to present computer science at the secondary school level in a way that would be both accessible and worthy of math/science credit, and
3) to offer additional secondary-level computer science courses that will allow interested students to study it in depth and prepare them for entry into the work force or college.

Before discussing the model curriculum itself, we first clarify the context in which it is set. Here, we would especially like to clarify the distinctions between computer science and information technology on the one hand, and summarize the nature of computer science at the college and university level on the other.

2.1 Computer Science, Information Technology, and Fluency

Information technology (IT) involves the proper use of technologies by which people manipulate and share information in its various forms - text, graphics, sound, and video. While computer science and IT have a lot in common, neither one is fully substitutable for the other. Similarly, *software engineering* (SE) is the practice of designing and implementing large software systems (programs). While computer science and SE have a lot in common, neither one of these is fully substitutable for the other.

A recent National Academy study [NAS99] defines an idea called *IT fluency* as something more comprehensive than IT literacy. Whereas IT Literacy is the capability to use *today's* technology in one's own field, the notion of IT Fluency adds the capability to independently *learn* and use *new* technology as it evolves [NRC99] throughout one's professional lifetime. Moreover, IT fluency also includes the active use of algorithmic thinking (including programming) to solve problems, whereas IT literacy is more limited in scope.

Thus, the field of *computer science* sits in a continuum - some of its topics overlap with IT, while some are completely different and would not be relevant to an IT curriculum. For example, the complexity of algorithms is a fundamental idea in computer science but would probably not appear in an IT curriculum. While IT is an applied field of study, driven by the practical benefits of its knowledge, computer science also has scientific and mathematical dimensions. Some of those dimensions, like the complexity of algorithms, expose the mathematical theory that underlies the practice, thus providing completeness to the study of computer science. Therefore, any comprehensive K-12 computer science curriculum will necessarily have topics that are distinct from those that normally appear in an IT curriculum.

[1] While programming is a central activity in computer science, it is only a tool that provides a window into a much richer academic and professional field. That is, programming is to the study of computer science as literacy is to the study of literature.

The idea of IT fluency [NAS99] was proposed as a minimum standard that all college students should achieve by the time they graduate. A "fluent" graduate would master IT on three orthogonal axes – *concepts*, *capabilities*, and *skills*.

The *concepts* are the ten basic ideas that underlie modern computers, networks, and information:
Computer organization, information systems, networks, digital representation of information, information organization, modeling and abstraction, algorithmic thinking and programming, universality, limitations of information technology, and societal impact of information technology.

The *capabilities* are the following ten fundamental abilities for using IT to solve a problem:
Engage in sustained reasoning, manage complexity, test a solution, manage faulty systems and softwrae, organize and navigate information structures and evaluate information, collaborate, communicate to other audiences, expect the unexpected, anticipate changing technologies, and think abstractly about IT.

The *skills* are the following ten abilities to use today's computer applications in one's own work:
Set up a personal computer, use basic operating system features, use a word processor and create a document, use a graphics or artwork package to create illustrations, slides and images, connect a computer to a network, use the Internet to find information and resources, use a computer to communicate with others, use a spreadsheet to model simple processes or financial tables, use a database system to set up and access information, and use instructional materials to learn about new applications or features.

Many colleges and universities (e.g., see [NAS99]) have implemented these or similar standards and are expecting their graduates to achieve them.

2.2 Computer Science at the College/University Level

Computer science is well developed at the college and university level. In the US alone, nearly every undergraduate college offers a major in computer science, and more than 100 universities offer PhD programs in computer science. Together, these programs produce about 24,000 baccalaureate and 900 PhD degrees each year [Taulbee2002].

The current model for college computer science major programs was published in 2001 [ACM/IEEE01]. This model identifies the following "core" subjects in 13 distinct areas that all computer science major programs should cover. Altogether, this material covers the equivalent of seven (7) one-semester courses, or 280 lecture hours (total lecture hours for each subject area are given in parentheses).

- *Algorithms and Complexity* (31): analysis of algorithms, divide-and-conquer strategies, graph algorithms, distributed algorithms, computability theory
- *Architecture* (36): digital logic, digital systems, data representation, machine language, memory systems, I/O and communications, CPU design, networks, distributed computing
- *Discrete Structures* (43): functions, sets, relations, logic, proof, counting, graphs and trees
- *Graphics and Visual Computing* (3): fundamental techniques, modeling, rendering, animation, virtual reality, vision
- *Human-Computer Interaction* (8): principles of HCI, building a GUI, HCI aspects of multimedia and collaboration
- *Information Management* (10): database systems, data modeling and the relational model, query languages, data mining, hypertext and hypermedia, digital libraries
- *Intelligent Systems* (10): fundamental issues, search and optimization, knowledge representation, agents, natural language processing, machine learning, planning, robotics
- *Net-centric Computing* (15): Introduction to Net-centric computing, the Web as a client-server example , network security, data compression, multimedia, mobile computing
- *Operating Systems* (18): concurrency, scheduling and dispatch, virtual memory, device management, security and protection, file systems, embedded systems, fault tolerance
- *Programming Fundamentals* (38): algorithms and problem-solving, fundamental data structures, recursion, event-driven programming
- *Programming Languages* (21): history and overview, virtual machines, language translation, type systems,

abstraction, OO programming, functional programming, translation
- *Social and Professional Issues* (16): ethical responsibilities, risks and liabilities, intellectual property, privacy, civil liberties, crime, economics, impact on the Internet
- *Software Engineering* (31): metrics, requirements, specifications, design, validation, tools, management

Undergraduate computer science programs also provide students with access to well-equipped computer laboratories and networks, since laboratory work is an essential component of the curriculum.

When computer science majors finish college, they have a number of capabilities. Some programs prepare graduates for advanced study, while others (the majority) prepare them for entry into the work force. For workforce entry, a graduate should [ACM/IEEE/01]:

1. Understand the essential facts, concepts, principles, and theories relating to computer science and software applications.
2. Use this understanding to design computer-based systems and make effective tradeoffs among design choices.
3. Identify and analyze requirements for computational problems, and design effective specifications.
4. Implement (program) computer-based systems.
5. Test and evaluate the extent to which a system fulfills its requirements.
6. Use appropriate theory, practice, and tools for system specification, design, implementation, and evaluation.
7. Understand the social, professional, and ethical issues involved in the use of computer technology.
8. Apply the principles of effective information management and retrieval to text, image, sound, and video information.
9. Apply the principles of human-computer interaction to the design of user interfaces, web pages, and multimedia systems.
10. Identify risks or safety aspects that may be involved in the operation of computing equipment within a given context.
11. Operate computing equipment and software systems effectively.
12. Make effective verbal and written presentations to a range of audiences.
13. Be able to work effectively as a member of a team.
14. Understand and explain the quantitative dimensions of a problem.
15. Manage one's own time and develop effective organizational skills
16. Keep abreast of current developments and continue with long-term professional growth.

It is not unreasonable to expect that a K-12 computer science program should expect its graduates to have some of these capabilities and skills as well.

2.3 The Current Status of K-12 Computer Science

Computer science is not widely taught at the K-12 level. To help address this problem, the ACM Model High School Curriculum [ACM93] was developed in 1993. This is a one-year course that covers core subjects, applications, and related topics.

The core topic selection in the 1993 model was motivated by an earlier, and now dated, college curriculum model than the one summarized above. That model included the study of Algorithms, Programming Languages, Operating Systems and User Support, Computer Architecture, and the Social and Ethical Context of computing. Its applications included CAD/CAM, speech, music, art, database, e-mail, multimedia and graphics, science and spreadsheets, word processing, and desktop publishing. Its electives included topics like AI (expert systems, games, robotics), computational science, simulation and virtual reality, software engineering, computational science, simulation and virtual reality, and software engineering.

For a variety of reasons, the ACM model curriculum was not widely implemented in secondary schools. One strong reason is that, since 1993, enormous changes have occurred in computer science itself, many of which were spurred

by the emergence of the World Wide Web. These changes have worked to accelerate the datedness of the core topics in the 1993 model.

A more recent curriculum model, developed by a New Jersey Teachers' Conference [Deek99], aimed to provide a state-level standard for computer science that could be taught in all school districts. The core topics for that curriculum include algorithms, programming, applications, information systems, communications, and technology. This curriculum is designed for use in the 9th, 10th, and 12th grades, in a way that complements the AP computer science curriculum (offered in the 11th grade). The 9th grade course provides an introduction to programming and problem solving, the Internet, information, communication, hardware, social impact and ethics; the 10th grade course emphasizes programming and applications. At the 12th grade level, a "topics" course provides an opportunity to offer interesting subjects like robotics, simulations, and animation.

In spite of these efforts, a recent survey (http://www.acm.org/education/k12/survey_hscs.html) suggests that neither the 1993 ACM model nor any other model has achieved widespread recognition or implementation in the United States. Seventy respondents, representing twenty-seven (27) states and three foreign countries, provided the following information.

Only 12 out of the 70 respondents replied that they have a state-mandated computer science curriculum at the high school level. However, the nature of that curriculum varied from state to state. The most extensive one identifies a separate computer science course at each grade level (9-12), while the most modest one designated "Introduction to the Computer" and "Internet Use of the Computer" as the only two state-mandated courses (at the grade 9 and 10 levels). So, even for states that offer computer science at all, there is much divergence in the number and content of these courses.

Where they are offered, computer science courses seem to be available only as electives (only one out of the 70 respondents indicated that computer science was mandatory).

As for teacher preparation and certification, 27 of the 70 respondents replied that their state requires no computer science certification to teach computer science courses. A different source notes that secondary computer science courses are usually taught by faculty certified to teach mathematics [Deek99].

On the international level, the development of K-12 computer science seems to be making slightly more headway than in the US.

In Israel, a secondary school computer science curriculum [Gal99] was approved by the Ministry of Higher Education and implemented in 1998. It blends conceptual and applied topics, and is offered in the 10th, 11th, and 12th grades. All students in the 10th grade are required to take a 1/2-year course in the foundations of computer science. This is followed by 1-1/2 or 2-1/2 years of electives taught at the 11th and 12th grades. These electives have a particularly heavy emphasis on the foundations of algorithms, which would be most appropriate for students planning to continue their education beyond secondary school.

In Canada, a comprehensive curriculum was recently implemented for all secondary schools in Ontario [Stephenson01]. It provides two alternative tracks, one emphasizing computer science and the other emphasizing computer engineering. All courses balance foundational knowledge with skills acquisition, and they prescribe outcomes at each level. At the 9th grade level, a full-year "integrated technologies" course is available to all students. This is followed by three parallel 3-year tracks – one in computer and information science and two in computer engineering.

3. A Comprehensive Model Curriculum

Building on the lessons of the past and the needs of the present and the future, we propose a four-level model curriculum for K-12 computer science that has the following general goals:

1. The curriculum should prepare students to understand the nature of computer science and its place in the modern world.

2. Students should understand that computer science interleaves principles and skills.
3. Students should be able to use computer science skills (especially algorithmic thinking) in their problem-solving activities in other subjects.
4. The computer science curriculum should complement IT and AP computer science curricula in any schools where they are currently offered.

If a K-12 computer science curriculum is widely implemented and these goals are met, high school graduates will be prepared to be knowledgeable users and critics of computers, as well as designers and builders of computing applications that will affect every aspect of life in the 21st century.

The overall structure of this model is shown in Figure 1. As this figure suggests, our model is delineated into four different levels, whose goals and content are summarized below.

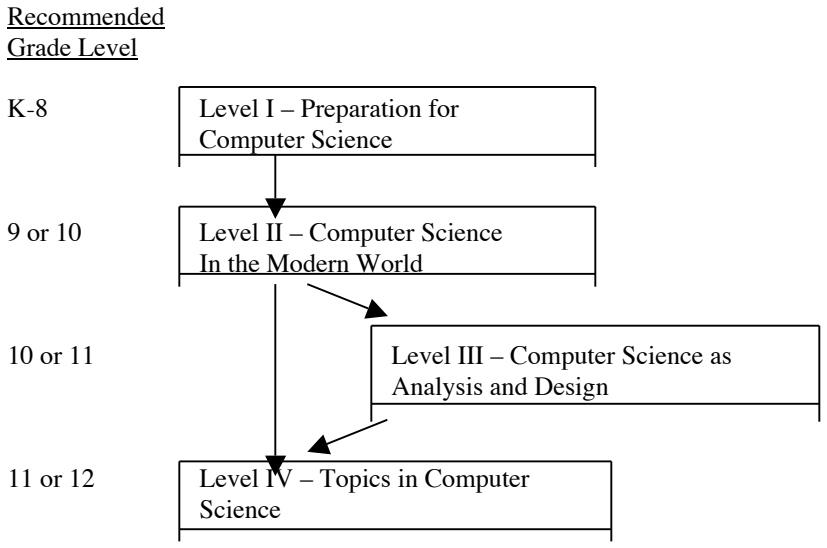

Figure 1. Structure of a K-12 Computer Science Curriculum

Level I (recommended for grades K-8) should prepare elementary school students for computer science by integrating basic skills in the use of technology with simple ideas about algorithmic thinking. This can be accomplished by adding short modules to existing science, mathematics, and social studies units. A combination of the NETS [NETS02] standards and an introduction to algorithmic thinking (as offered, for instance, by Logo [Papert] or programmable robots) would ensure that students meet this goal.

Students at *Level II* (recommended for grade 9 or 10) should acquire a coherent and broad understanding of the principles, methodologies, and applications of computer science in the modern world. This can be offered as a one-year course accessible to all students, whether they are college-bound or workplace-bound. For most students, this will be their last encounter with computer science.

Students who wish to study more computer science may elect the *Level III* (recommended for grade 10 or 11) course, a one-year elective that would earn math or science credit. This course continues the study begun at Level II, but it places particular emphasis on the scientific and engineering aspects of computer science – mathematical principles, algorithmic problem solving and programming, software and hardware design, networks, and social impact. Students will elect this course to explore their interest and aptitude for computer science as a profession.

Finally, the *Level IV* (recommended for grade 11 or 12) offering is an elective that provides depth of study in one particular area of computer science. This may be, for example, an AP computer science [APCS02] course, which offers depth of study in programming and data structures. Alternatively, this offering may be a projects-based course in multimedia design or a vendor-supplied course that leads to professional certification. Any *Level IV* course will naturally require the *Level 2* course as a prerequisite, and some will require the *Level 3* course as well.

The following subsections present more detailed descriptions of the topics and courses that can be offered at each of these four levels.

3.1 Level I - Preparation for Computer Science

Because preparation for computer science has a major information technology component, it is important here to reaffirm the need for technology support in the K-12 classroom.[2] Successful integration of technology to support learning goals depends upon several factors:

- vision and leadership for successful implementation and long-term success,
- access to physical resources (hardware and software),
- physical arrangement of those resources in accessible learning spaces,
- time and incentives to support classroom-relevant professional development opportunities for educators,
- time for planning effective integration into new and existing curricula,
- time for reviewing and evaluating new technologies and resources, and
- ongoing financial support for a sustained technology infrastructure.

It also depends upon a clear vision of what expectations are necessary and appropriate at every level. In this document we explore a number of different levels of computer science education throughout the K-12 years. It is clear to us that whatever is achieved in high school depends upon the effectiveness of student access to technology and achievement of computer-related learning milestones at the elementary level. So if elementary schools provide students with these first building blocks of computer fluency, secondary schools will be able to implement more comprehensive computer science programs themselves.

NETS Standards

The National Educational Technology Standards (NETS) [ISTE02] provide an excellent starting place for defining requirements for elementary student preparedness in computer science.[3]

In order to live and work successfully in an increasingly information-rich society, K-8 students must learn to use computers effectively and incorporate the idea of algorithmic thinking into their daily problem-solving vocabulary. To these ends, students should become:

- Computer users,
- Information seekers, analyzers, and evaluators,
- Aware that some problems have multi-step algorithmic solutions,
- Problem solvers and decision makers,
- Users of productivity tools, and
- Information producers and communicators.

---

[2] Too frequently, new and complex expectations are downloaded onto classroom teachers without a realistic consideration of the resources available to teachers to achieve these expectations. Often, there is an assumption that technology itself is the panacea, and so, little consideration is giving to preparing teachers to use the technology effectively and in support of their own teaching and learning goals.

[3] These standards were originally developed by the International Society for Technology in Education (ISTE) as part of an on-going effort to enable stakeholders in Pre-K-12 education to develop national standards for educational uses of technology.

To ensure these outcomes, schools must provide computing tools that enable students to solve problems and communicate using a variety of media; to access and exchange information; compile, organize, analyze, and synthesize information; draw conclusions and make generalizations from information gathered; understand what they read and locate additional information as needed; become self-directed learners; collaborate and cooperate in team efforts; analyze a problem and develop an algorithmic solution; and interact with others using computers in ethical and appropriate ways.

Topics and Goals

To prepare K-8 students for computer science, the following topics and performance goals are identified.

Basic operations and concepts Students demonstrate a sound understanding of the nature and operation of computers, and are proficient in their use.

Social, ethical, and human issues Students understand the ethical, cultural, and societal issues related to computers; practice responsible use of computers, information, and software; and develop positive attitudes toward computer uses that support lifelong learning, collaboration, personal pursuits, and productivity.

Productivity tools Students use tools to enhance learning, increase productivity, and promote creativity; and use productivity tools to collaborate in constructing models, prepare publications, and produce other creative works.

Communication tools Students use telecommunication to collaborate, publish, and interact with others; and use different media and formats to exchange information and ideas effectively with various audiences.

Technology research tools Students use computers to locate, evaluate, and collect information from a variety of sources; use technology tools to process data and report results; and evaluate and select new information resources and technological innovations based on the appropriateness for specific tasks.

Technology problem-solving and decision-making tools Students use technology resources for solving problems and making informed decisions; and employ technology in the development of strategies for solving problems in the real world.

Algorithmic problem-solving tools Students use appropriate programming languages, simulators, and robots to discover and analyze multi-step algorithmic solutions to problems.

Grade-Level Breakdowns

To ensure that students achieve these goals, we recommend the NETS model, which identifies different sets of outcomes for three different groups of students: grades K-2, grades 3-5, and grades 6-8.

Grades K-2: Upon completion of Grade 2, students will:

1. Use standard input and output devices to successfully operate computers and related technologies.
2. Use a computer for both directed and independent learning activities.
3. Communicate about technology using developmentally appropriate and accurate terminology.
4. Use developmentally appropriate multimedia resources (e.g., interactive books, educational software, elementary multimedia encyclopedias) to support learning.
5. Work cooperatively and collaboratively with peers, teachers, and others when using technology.
6. Demonstrate positive social and ethical behaviors when using technology.
7. Practice responsible use of technology systems and software.
8. Create developmentally appropriate multimedia products with support from teachers, family members, or student partners.
9. Use technology resources (e.g., puzzles, logical thinking programs, writing tools, digital cameras, drawing tools) for problem solving, communication, and illustration of thoughts, ideas, and stories.
10. Gather information and communicate with others using telecommunications, with support from teachers, family members, or student partners.

Grades 3-5: Upon completion of Grade 5, students will:

1. Be comfortable using keyboards and other input and output devices, and reach an appropriate level of proficiency using the keyboard with correct fingering.)
2. Discuss common uses of technology in daily life and the advantages and disadvantages those uses provide.

3. Discuss basic issues related to responsible use of technology and information, and describe personal consequences of inappropriate use.
4. Use general-purpose productivity tools and peripherals to support personal productivity, remediate skill deficits, and facilitate learning throughout the curriculum.
5. Use technology tools (e.g., multimedia authoring, presentation, Web tools, digital cameras, scanners) for individual and collaborative writing, communication, and publishing activities to create presentations for audiences inside and outside the classroom.
6. Use telecommunications efficiently to access remote information, communicate with others in support of direct and independent learning, and pursue personal interests.
7. Use online resources (e.g., e-mail, online discussions, Web environments) to participate in collaborative problem-solving activities for the purpose of developing solutions or products for audiences inside and outside the classroom.
8. Use technology resources (e.g., calculators, data collection probes, videos, educational software) for problem solving, self-directed learning, and extended learning activities.
9. Determine which technology is useful and select the appropriate tool(s) and technology resources to address a variety of tasks and problems.
10. Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and bias that occurs in electronic information sources.

Grades 6-8: Upon completion of Grade 8, students will:

1. Apply strategies for identifying and solving routine hardware and software problems that occur during everyday use.
2. Demonstrate knowledge of current changes in information technologies and the effect those changes have on the workplace and society.
3. Exhibit legal and ethical behaviors when using information and technology, and discuss consequences of misuse.
4. Use content-specific tools, software, and simulations (e.g., environmental probes, graphing calculators, exploratory environments, Web tools) to support learning and research.
5. Apply productivity/multimedia tools and peripherals to support personal productivity, group collaboration, and learning throughout the curriculum.
6. Design, develop, publish, and present products (e.g., Web pages, videotapes) using technology resources that demonstrate and communicate curriculum concepts to audiences inside and outside the classroom.
7. Collaborate with peers, experts, and others using telecommunications tools to investigate educational problems, issues, and information, and to develop solutions for audiences inside and outside the classroom.
8. Select appropriate tools and technology resources to accomplish a variety of tasks and solve problems.
9. Demonstrate an understanding of concepts underlying hardware, software, algorithms, and their practical applications.
10. Discover and evaluate the accuracy, relevance, appropriateness, comprehensiveness, and bias of electronic information sources concerning real-world problems.

The Case for Algorithmic Problem Solving

Except in the context of mathematics education, this particular topic area is not a conventional part of the K-8 curriculum. That is, the concept of algorithm is used only to teach students the steps of arithmetic (addition, multiplication) and other basic mathematical ideas. However, the notion of algorithm affects students in a much richer array of problem solving situations that they encounter in their lives.

In its simplest form, an algorithm is a symbolic representation of a method for solving a problem in a step-by-step manner. Essentially, children learn about problem solving as developing a collection of steps that must be carried out in a particular sequence to accomplish a task. These steps must accommodate unusual contingencies (using conditional, or "if" statements) and repetitions (using loops, or "while" statements). Viewed in this way, algorithmic thinking is not simply a means to help children understand mathematical concepts - it has a much richer range of uses. Here are a few example problems that illustrate this point and would be appropriate at the K-8 level.

Give a complete algorithmic definition for ...

1. … finding your way out of a maze (Turtle graphics, robotics),
2. … a dog retrieving a thrown ball,
3. … baking cookies,
4. … going home from school,
5. … making a sand castle,
6. … arranging a list of words in alphabetical order.

Thus, we agree with the minority of teachers and others who believe that students at this age ought to begin thinking algorithmically as a general problem solving strategy. What children do, not what they see, may have the greatest impact on learning at the K-8 level. Thus, it makes sense to develop more teaching strategies that get students maximally involved in the process of visualizing an algorithm. Seymour Papert's pioneering experiments in the 1980s corroborate this belief, and his seminal work *Mindstorms* and related curricula [Papert] provide many more examples of how K-8 students can be engaged in algorithmic thinking.

3.2 Level II – Computer Science in the Modern World

This is a year-long course (or equivalent) that would be accessible to all students, whether they are college-bound or workplace-bound. The goal of this course is to provide all students with an introduction to the principles of computer science and its place in the modern world. This course should also help students to use computers effectively in their lives, thus providing a foundation for successfully integrating their own interests and careers with the resources of a technological society.

In this course, high school students can acquire a fundamental understanding of the operation of computers and computer networks and create useful programs implementing simple algorithms. By developing web pages that include images, sound, and text, they can acquire a working understanding of the Internet, common formats for data transmission, and some insights into the design of the human-computer interface. Exposure to career possibilities and discussion of ethical issues relating to computers should also be important threads in this course.

Prior to this course, students should have gained experience using computers, as would normally occur at Level I. They should have used, modified, and created files for a variety of purposes, accessed the Internet and databases for both research and communication, and used other tools such as spreadsheets and graphics. Finally, they should have been introduced to the basic idea of algorithmic thinking and its uses in their daily lives.

Topics and Goals

A major outcome of this course is to provide students with general knowledge about computer hardware, software, languages, networks, and their impact in the modern world.[4] For instance, the idea that a robot needs a method of acquiring sensory data from its environment draws attention to the general notion of an "input device" beyond the standard keyboard and mouse. Learning about various input devices currently in use should help demystify the general idea of input, and prepares them to be comfortable with using devices with which they are not yet familiar.

Students should gain a conceptual understanding of the following "great ideas" in computer science:

1. Principles of computer organization and the major components (input, output, memory, storage, processing, software, operating system, etc.)
2. The basic steps in algorithmic problem solving (problem statement and exploration, examination of sample instances, design, program coding, testing and verification)
3. The basic components of computer networks (servers, file protection, queues, routing protocols for connection/communication, spoolers and queues, shared resources, and fault-tolerance)

[4] Coincidentally, students will acquire proficiency with a current computer model and programming language, but that is not the main goal of this course.

4. Organization of Internet elements, Web page design (buttons, menus, text areas, graphics, concepts of applets and scripts, client-server computing), and hypermedia (links, navigation, search engines and strategies, interpretation, and evaluation).
5. The notion of hierarchy and abstraction in computing, including high-level languages, translation (compilers, interpreters, linking), machine languages, instruction sets, and logic circuits.
6. The connection between elements of mathematics and computer science, including binary numbers, logic, sets, and functions.
7. The notion of computers as models of intelligent behavior, as found in robot motion, speech and language understanding, and computer vision.
8. Examples (like programming a telephone answering system) that identify the broad interdisciplinary utility of computers and algorithmic problem solving in the modern world.
9. Ethical issues that relate to computers and networks, including security, privacy, intellectual property, the benefits and drawbacks of public domain software, and the reliability of information on the Internet.
10. Identification of different careers in computing and their connection with the subjects studied in this course (E.g., information technology specialist, Web page designer, systems analyst, programmer, CIO).

Laboratory work: Algorithms, Programming, and Web Page Design

Students in this course should gain experience designing algorithms and programming solutions to a variety of computational problems. While the choice of programming language and environment is up to the instructor, the programming component of the course should include the following:

- Variables, data types, and the representation of data in computers
- Managing complexity through top-down design
- Procedures and parameters
- Sequences, conditionals, and loops (iteration)
- Tools for expressing design – flowcharts, pseudocode, UML

The Web page design component of this course should cover the following ideas:

- The use of links to load new pages or activate processes
- Formats for storing images and sound
- Techniques for compressing and encrypting data
- Formats for Web page reference (URL and URI)
- Relative versus global referencing of files
- Options for user interfaces
- Tools for expressing design – storyboard, site map

Context and Constraints

Each school system has its own constraints with regard to student scheduling, availability of knowledgeable staff and computer resources. Some schools may choose to begin by implementing an elective course that covers only a subset of the above concepts. We believe that, while such initial steps are valuable, they must nonetheless be identified as first steps towards the ultimate goal of a full course required of all students for graduation.

Finally, it is important to distinguish the goals and themes of this course from those of information technology, especially those that comprise the notion of "IT fluency" (see section 2). This course provides the first opportunity to view computer science as a coherent field of study and professional engagement. That is, while IT fluency focuses on technological skills and their applications in other subjects and professions, this course is a study of computer science *per se*.

3.3 Level III – Computer Science as Analysis and Design

This is a year-long course (or equivalent) that should earn science or math credit. The goal of this course is to continue the study of computer science, placing particular emphasis on its features as a scientific and engineering discipline.

In this course, high school students can go beyond a fundamental understanding of the operation of computers and explore more complex and interesting topics of computer science. This course also helps students improve their problem-solving and programming skills in preparation for the Advanced Placement A course. As in higher level math and science curricula, students will be able to see the connection between the fundamentals they have learned in Levels I and II in order to integrate programming and design with complex "real world" projects.

Topics and Goals

The major goal of this course is for students to develop the computer science skills of algorithm development, problem solving, and programming while using software engineering principles. While the emphasis of the course will be on programming, students will also be introduced to other important topics, such as interface design, the limits of computers, and societal and ethical issues of software engineering.

By the end of this course, students should understand or have a working knowledge of these topics:

1. Fundamental ideas about the process of program design and problem solving, including style, abstraction, and initial discussions of correctness and efficiency as part of the software design process.
2. Simple data structures and their uses
3. Design for usability - Web page design, interactive games, documentation
4. Fundamentals of hardware design
5. Levels of language, software, and translation: characteristics of compilers and operating systems
6. The limits of computing: what is a computationally "hard" problem? (e.g., ocean modeling, air traffic control, gene mapping) What kinds of problems are computationally unsolvable? (e.g., the halting problem)
7. Principles of software engineering: software projects, teams, the software life cycle
8. Social issues: Software as intellectual property, professional practice
9. Careers in computing: computer scientist, computer engineer, software engineer, information technologist

Laboratory Work: Programming, Design, and Other Activities

Students in this course should gain experience designing algorithms and programming solutions to a variety of computational problems. While the choice of programming language and environment is up to the instructor, the programming component of the course should include the following:

- Methods (functions) and parameters
- Recursion
- Objects and classes (arrays, vectors, stacks, queues, and their uses in problem solving)
- Graphics programming
- Event-driven and interactive programming

Hardware and software engineering has several topics that can be introduced during this course and included among its programming projects:

- Hardware and systems: logic, gates and circuits, binary arithmetic, machine and assembly language, operating systems, user interfaces, compilers
- Software engineering: requirements, design, teams, testing and maintenance, documentation, software design tools
- Societal issues in software engineering, limits of computing, levels of languages, computing careers

Context and Constraints

Since this is a laboratory-intensive course, students will need regular access to appropriate computing facilities and software. A number of viable alternative programming language alternatives exist, and so we recommend no particular programming language to support this course. Surely, the choice of language depends on local conditions, such as teacher expertise, laboratory hardware configuration, and availability and cost of software support.

Moreover, this course is intended to be much broader in scope than the AP curriculum, and thus should complement it in a way that is accessible to all students… not just those preparing for college. However, for students who are thinking about taking an AP computer science course at Level IV (see below), this course can serve as a precursor.

This course is also intended to cover the fundamentals of computer science more broadly than a typical information technology course. While it has elements of IT, this course also introduces students to concepts that are not typically covered in and IT curriculum, such as the limits of computing and data structures.

3.4 Level IV – Topics in Computer Science

At this level, interested and qualified students should be able to select one from among several electives to gain depth of understanding or special skills in particular areas of computer science. All of these electives will require the Level 2 course as a prerequisite, while some may require the Level 3 course as well. Most importantly, these courses provide students with an opportunity to explore topics of personal interest in greater depth, and thus prepare themselves for the workplace or for further study at the post-secondary level.

These electives include, but are not necessarily limited to:

- Advanced Placement Computer Science [APCS02]
- A projects-based course in which students cover a topic in depth.
- A vendor-supplied course, which may be related to professional certification.

These are discussed in more detail below.

AP Computer Science

The AP computer science curriculum is well established, and is offered at many secondary schools for students planning to continue their education in a 2- or 4-year college or university, possibly in computer science, business, or a related field.

Students taking an AP course should have completed Levels 1 and 2. Students entering an AP Computer Science course need to be familiar with the basic algorithmic concepts introduced at those levels. The programming concepts covered in Level 3 overlap somewhat with the AP course, so some of the AP course can serve as a review if students have had the Level 3 course.

The curriculum that prepares students for the AP computer science exams provides an excellent foundation for future study. This curriculum has two courses:

- The A course emphasizes problem solving and algorithm development, and it also introduces elementary data structures. Students who complete the A course and score well on the exam may qualify for one-semester of college credit.
- The AB course extends the foundation of the A course by including more substantial work with data structures and recursive algorithms.

The College Board suggests that the choice between A versus AB be left to the school and students. A school might wish to initially offer the A course as it is less comprehensive, and then move toward the AB course as instructor knowledge and entering student level increases.

In schools that implement this curriculum recommendation, students will arrive at Level 4 with a standard background that enables them to be successful in the AB Course. Also, high schools need to consider the significant staffing issues implied by this curriculum recommendation, along with the staffing tradeoffs that result from offering 0, 1, or 2 AP courses in a setting that also offers the Level 2 and 3 courses described above.[5] For example, a school

[5] Achieving a high score on the AP A Exam is typically considered to be equivalent to completing a one-semester college course in computer science. Programming language differences between the AP exam and the one taught at

that is neither large nor resource-rich may prefer to offer the Level 3 course alone, and then supplement that course with additional material that will support a smaller group of students preparing for the AP A exam.

Projects-Based Courses

This kind of course would be available to all students who have completed the Level 1 and 2 curricula. Some variants of this course would also require completion of Level 3 (see below). This could be either a 1/2-year or a full-year course.

The projects in this kind of course will naturally address diverse student interests and specific faculty expertise. The specific projects that are chosen from year to year will be fluid and will adjust as needed to meet the ever-changing characteristics of computer science and information technology. Ideally, each project should build upon basic computer science concepts and help students develop professional skills in the application of technology.

While some of the project curriculum may be more skills-based, the skills need to be tied to the "behind-the-scenes" activities of the software – particularly how is each task implemented in the software (e.g., what is happening when you click "bold"?). Answering such questions enables students to problem-solve when software does not perform as anticipated. Additional computer science topics are visited throughout these projects.

Here are some example projects that could populate such a course.

Example: Desktop Publishing This course introduces planning, page layout, and the use of templates to create flyers, documents, brochures, and newsletters. Word processing and graphical editing fluency (Level 1) will help insure student success. Methods of distribution of these documents in both written and electronic formats should be included. This will necessitate understanding of Internet concepts and network connectivity (Level 2).

Example: Presentation Design The ability to communicate and share ideas should be a core requirement for all high school graduates. Communication can be written and/or oral. This type of project focuses on planning a presentation – including outlining, converting the outline into a document, and generating the presentation. Concepts covered include appropriate use of text, colors, graphics, sound, and animations on slides as well as linking within and outside the presentation. Ultimately, students will present to an audience. Fluency with word processing software (Level 1) and multimedia concepts (Level 2) is required.

Example: Multimedia The use of multimedia is increasing steadily at the user level, fueled by more efficient hardware and the availability of digital cameras and digital audio equipment. However, multimedia is often abused when incorporated into programs, web pages, and presentations. This project will provide instruction in the use of digital audio and video equipment and related editing software. A major focus will be deploying multimedia in a responsible fashion. Basic software skills (Level 1) and an understanding of multimedia concepts (Level 2) are required.

Example: Graphics This class explores bitmap and vector-based graphics. The discussion includes benefits and limitations of each type of software and hands-on experience with both. CAD, CAM, and 3-D design software should be explored as well as bitmap software for creation and editing of graphics. Availability of a digital camera and scanner is required. Responsible deployment of graphics including style and legal issues needs to be investigated. The discussion of vector-based graphics will be facilitated by completion of Level 3 – limits of computers and design for usability.

Example: Design and Development of Web Pages At Level 2, students are exposed to Internet concepts and HTML. This course presents a more in-depth view of the design and development issues that need to be considered for a multi-platform international implementation. A focus issue is the standardization of web page development

---

a particular college (e.g., C++ vs. Java) may present an issue in granting AP credit for students with high scores. That is, some colleges may require students to repeat the introductory semester(s) so they can continue effectively in the undergraduate computer science major program.

using the recommendations of the WWW Consortium. Web page development is presented and evaluated using text editors, HTML editors, converters, and web authoring programs.

Example: Web Programming Students who have successfully completed Level 3 but do not wish to take an AP course might nevertheless enjoy applying their programming skills to the WWW. To be successful, a solid understanding of Internet concepts, web page design and development issues, and basic programming concepts will be required. Topics in this course can include client-side and server-side scripting languages. Students will need to write scripts and deploy them within web pages or on the web server.

Example: Emerging Technologies This project can include several distinct topics, and its content is expected to change on a regular basis. An example topic for upcoming years might be XML/XSL and wireless connectivity. These areas can be tied together with a discussion of requirements for the same data to be represented on a PC, personal digital assistant (PDA), and cell phone. Curriculum and materials for this topic would need to be developed from current resources on the Web, perhaps in conjunction with local colleges and universities, and with input from the professional sector of the Business Community.

A sample of some other topics (along with their prerequisites) is:

- The Computer and Animation (Level II)
- Networking Technologies (Level III)
- Programming simulations (Level III)
- Object-oriented design and coding (Level IV – AP computer science)
- Effective use of Computer Applications (Level II)

Courses Leading toward Industry Certification

Such a course is primarily geared toward students planning on entering the workforce, continuing their education in a post-secondary technical school, or entering a 2-year college AAS program. Students taking this course should have completed the Level 1 and Level 2 courses.

Industry certification provides a standard that is useful to potential employers in evaluating a candidate who has no prior work experience. Industry certifications are either vendor-neutral or vendor-sponsored. Vendor-sponsored curricula need to be evaluated carefully. While rich in content, some of these courses are structured to emphasize proprietary products rather than general concepts. Students who complete certification courses should be encouraged to take the corresponding exam as proof of acquired knowledge. Here are some examples of vendor-neutral certification programs.

Example: A+ Certified Technician "A+ certification signifies that the certified individual possesses the knowledge and skills essential for a successful entry-level (6 months experience) computer service technician, as defined by experts from companies across the industry." (http://www.comptia.org/certification/a/default.asp). Two different exams are available – software and hardware. Both of these assume that students have gained an understanding of the way a computer works, including hands-on experience. The hardware section includes installation of new equipment and troubleshooting. The software section encompasses various operating systems. The use of critical thinking skills to problem-solve is necessary for hardware and software support. These skills reinforce and extend the concepts presented in Level 1 and 2. The A+ exam is vendor-neutral.

Example: Certified Internet Webmaster (CIW) The CIW exams are also vendor-neutral. "CIW certification validates competency in IT industry standards, concepts and best practices; and familiarity with leading hardware and software technology." (http://www.ciwcertified.com/program/about.asp?comm=home&llm=1) The Foundations level exam requires competency in Internet, web page authoring, and networking fundamentals. These concepts are introduced in Level 1 and 2. While the scope of the exam is beyond the reach of high school students, its objectives can serve to extend the foundation of the previously discussed related issues.

Example: i-Net+ This certification is designed for "individuals interested in demonstrating the baseline of technical knowledge that would allow them to pursue a variety of Internet-related careers. The i-Net+ exam was specifically designed to certify entry-level Internet and e-commerce technical professionals responsible for participating in the

maintenance of Internet, Intranet and Extranet infrastructure and services as well as the development of Web-related applications." (**http://www.computer-certification-training.com/CompTIA/inet/i-net.html**).

More detailed information about these and other certification programs, both vendor-specific and vendor-neutral, can be found at the Web site http://www.computer-certification-training.com/index.html.

4. Implementation Challenges

Teaching any subject effectively depends on the existence of a sound curricular model, explicit teacher certification standards, appropriate teacher training programs, and effective curricular materials. K-12 computer science education faces unique challenges along these lines because the subject is young and it has remained outside the mainstream K-12 curriculum. We believe that computer science education now needs to enter the mainstream.

For this to happen, and for schools to widely implement this model, work is needed in three important areas: *teacher preparation*, *state-level content standards, and curriculum materials development*. As indicated earlier, some states have begun to establish content standards, define models for teacher certification, provide in-service training in computer science, and experiment with developing new curricular materials. However, a much wider effort and commitment are now required.

Wide adoption of K-12 computer science will be a difficult task. Professional organizations in computer science can facilitate this task. Organizations that can participate in this effort include the Association for Computing Machinery (ACM), the International Society of Technology in Education (ISTE), institutions of higher education, as well as national and local teacher organizations.

Below is a discussion of the main challenges as we see them.

Teacher Preparation Standards

In order for students to master this new subject, teachers must acquire both a mastery of the subject matter and the pedagogical skills that will allow them to present the material to students at appropriate levels. It is understood that there must be a match between the computer science skills and knowledge defined for the students and the acquired skills and knowledge of the teachers. At the same time, teachers must have a greater depth of knowledge than that embodied in the topics they are teaching.

State departments of education and other appropriate agencies must recognize the discipline of computer science, so that appropriate standards for teacher certification are established. This should be followed by the establishment of teacher preparation programs with a prescribed course of study in computer science and education, so that perspective teachers will gain the skills and knowledge necessary to meet the certification standards required.

Due to an absence of standards, teachers graduating from colleges of education have not typically been well prepared to teach computer science. This issue has recently been addressed by the National Council for Accreditation of Teacher Education (NCATE), a coalition of thirty-three specialty professional associations of teachers, teacher educators, content specialists, and local and state policy makers. NCATE oversees the professional accreditation of schools, colleges, and departments of education. The NCATE policy boards develop NCATE standards, policies, and procedures. Currently, 525 institutions are accredited and another 100 are candidates and pre-candidates for accreditation. The number of candidates for accreditation has almost tripled in the past five years, due to the growing demand for accountability from states and the public.

The NCATE accreditation system is a voluntary peer review process that involves a comprehensive evaluation of the institution that prepares teachers and other professional school personnel. The review itself is based on the NCATE Unit Standards, which are developed by all sectors of the teaching profession. Accreditation requires an on-site review of the unit and a review of the individual programs within the unit.

NCATE has recently defined accreditation standards for secondary computer science education programs. It is anticipated that these standards can be implemented through a teacher preparation *endorsement* program, roughly equivalent to providing prospective teachers with a minor in computer science (including at least 18 semester hours of college-level computer science). The prerequisite for this program is a foundation in educational technology.

The NCATE accreditation standards for secondary computer science education programs use a definition of computer science that reflects the core requirements for college computer science majors [ACM/IEEE01]. These standards include programming and algorithm design, computer system organization and operation, data

representation and information organization, and social aspects of computing. Secondary school teachers certified by the NCATE standards must demonstrate the following specific computer science knowledge:

1. knowledge and skill regarding the syntax and semantics of a high level programming language, its control structures, and its basic data representations
2. knowledge and skill regarding common data abstraction mechanisms (e.g., data types or classes such as stacks, trees, etc.)
3. knowledge and skill regarding program correctness issues and practices (e.g., testing program results, test data design, loop invariants)
4. design and implementation of programs of sufficient complexity to demonstrate knowledge and skills
5. design, implement, and test programs in languages from two different programming paradigms in a manner appropriate to each paradigm
6. effective use of a variety of computing environments (e.g., single- and multi-user systems and various operating systems)
7. operation of a computer system--CPU & instruction cycle, peripherals, operating system, network components, and applications--indicating their purposes and interactions among them
8. machine level data representation (e.g., character, boolean, integer, floating point)
9. applications of the various data and file structures provided by a programming language (e.g., objects, various collections, files)
10. elements (people, hardware, software, etc.) in information systems (database systems, the Web, etc.) and their interactions
11. social issues related to the use of computers in society and principles for making informed decisions regarding them (e.g., security, privacy, intellectual property, limits of computing, rapid change)
12. significant historical events relative to computing
13. independent learning on other topics in computer science, including written and oral reports
14. participation in team software development projects that apply sound software engineering principles

By NCATE standards, computer science teachers must also possess the following capabilities.

1. Identify resources, strategies, activities, and manipulatives appropriate to teaching secondary computer science
2. Plan lessons/modules/courses related to each of: programming process and knowledge/concepts, and issue examination
3. Develop assessment strategies appropriate to lesson goals and the need to provide student feedback
4. Perform course and lesson planning that addresses student population characteristics (e.g., academic ability, cultural experience)
5. Observe and discuss the teaching of secondary computer science
6. Participate in the teaching of secondary computer science (lab assistant, tutoring, mini-teaching, etc.)
7. Plan and deliver a unit of instruction
8. Plan direct instruction involving simultaneous use of computing facilities by students (e.g., holding class in the lab, closed labs)
9. Plan instruction involving students independently using computing facilities
10. Develop a personal plan for evaluating their own practice of teaching
11. Make use of their plan for self-evaluation in the instructional delivery activities
12. Discuss guidance roles and possible enrichment activities for secondary computer science students (e.g., computing career guidance, preparation for college, and extracurricular activities such as computer clubs and organized competitions)
13. Plan for professional growth after identifying professional computer science and computer science education societies, organizations, groups, etc. that provide professional growth opportunities and resources

The development of teaching certification requirements and content standards for K-12 computer science education by the various states will in turn prompt the schools to implement relevant computer science programs. But most importantly, this step will also motivate schools of education to introduce pre-service programs in computer science education. With computer science becoming a recognized academic discipline in the schools, schools of education will become more motivated to set up such pre-service programs.

Some states already offer certification or an endorsement for teaching computer science. But the majority of states do not require any computer science credentials for teaching this subject. For those states that offer teachers an endorsement in computer science, their requirements vary widely - some require teachers to have a background in data processing while others require them to have a business background. Another concern is that some states' endorsements cover a very narrow aspect of computer science while others combine the subject with technology education for the purpose of certification.

The states' departments of education should consider reviewing their licensing standards for professional educators so that they recognize computer science as a new discipline. The meaning of the term "teacher of computer science" requires clear definition. As the requirements for certification and pre-service programs are developed, they must maintain the view that the field of computer science is rapidly evolving.

As with other subjects, in-service education is important to help current teachers adopt and integrate new computer science curriculum elements. In-service programs must deliver the needed professional development for the educators who will teach these courses. Provisions must be made to retrain teachers already in the school systems, so that they may also develop those skills and knowledge necessary to obtain new certifications as needed.

In-service education at the early stages of this curriculum implementation can take many forms. In addition to school- and district-wide workshops, state and regional events can be organized to bring teachers together as a community to learn and exchange ideas. These events can be used to disseminate to the teachers and school administrators new curricular recommendations and guidelines as they evolve. Another important goal of such events would be to provide short workshops regarding timely issues in computing for the preparation of the new curriculum implementation.

Professional recognition is important for the current cohort of teachers of computer science, regardless of the nature of their original teaching certification. Almost all of these teachers have original credentials in mathematics, science, business, and English, but they have since self-educated in order to teach many different types of computing courses, including AP computer science. One way to provide such recognition is to develop standard core competencies for computer science teachers and issue endorsements to the certificates of those teachers with these competencies, thus recognizing teachers who have the requisite skills and knowledge in computer science. This can be accomplished through new in-service training initiatives.

State-Level Content Standards

Recently, efforts have increased to develop national and state content standards for computer science. The curriculum standards serve to define the skills and knowledge of the discipline to be acquired by every student. For this to happen, school curricula must be aligned with these standards. Content standards for computer science education need to be developed and adopted in a way that parallels what has occurred in disciplines such as science, mathematics, and language arts. Curriculum frameworks aligned with these content standards can then be developed for the classroom.

In the design of state standards, it is important to ensure distinction between the teaching of IT skills (especially in service to the sciences and mathematics) and the teaching of computer science itself. That is, computer science must be considered as subject matter and technology should be viewed as a tool that cuts across all subjects. Existing technology standards, where present, should not be substituted for computer science standards.

Curriculum Development

This report presents a model for computer science education, but not a complete "deliverable" curriculum. Additional steps need to be taken to formulate content standards, define professional development needs, develop curriculum (textbooks and laboratory materials), and disseminate to students in the classroom. For all this to happen, teachers must play a substantial and leading role in the formulation of curriculum components. This will also require the participation of university faculty and professional organizations (ACM and ISTE) to serve as facilitators and guide a process that will yield a deliverable and effective curriculum.

One possible vehicle for mobilizing these efforts would be to seek grant support from federal agencies (e.g., NSF) and private foundations. Ideally, a summer institute in K-12 computer science education could be established and teachers would be chosen to participate in the development of curriculum content and teaching modules. The institute will made up of working groups and will be held at multiple locations throughout the country for two to three weeks each summer. Participants would come together the following summer to discuss results and plan follow-up activities.

Implementation and Sustainability

This report proposes a model, but not a "deliverable" curriculum in the form of teaching materials, lesson plans, a trained teaching cohort, or an operational budget to deliver K-12 computer science in the way suggested above. Additional steps are needed to begin this process of implementation in K-12 schools. The following are essential:

Buy-in - these recommendations should be endorsed widely by organizations that have a stake in their implementation: ACM SIGCSE, ISTE Special Interest Group for Computer Science (SIGCS), directors of curriculum in school districts (ASCD), state boards of education, National Education Association (NEA), National Association of Secondary School Principals (NASSP), National School Board Association.

Curriculum and course development - Funding sources like NSF should be approached to assist teams of K-12 teachers and other computer science educators to develop pilot courses along the lines suggested in this report. Concurrently, textbook and Web-based publishers should be encouraged to invest in these experimental courses, so that the resulting teaching materials can be widely disseminated and used elsewhere.

Professional societies - Establishment of a "National Council for Teachers of Computer Science," a new professional society for K-12 computer science teachers, should be considered. Similarly, ACM SIGCSE and NECC should continue to broaden their missions and conferences to better accommodate K-12 computer science teachers. State and regional organizations should provide on-going support and collaboration for K-12 computer science teachers at the local level.

Dissemination is a critical first step to implementation. Follow-up through local and regional organizations and national forums will further the implementation of these recommendations. Such events will provide opportunities for sharing and discussion of successful implementations, as well as for the discussion of problems encountered. In addition, these events will help further the recognition of computer science as an appropriate and necessary discipline for a comprehensive K-12 curriculum.

Additional steps will still be needed to sustain this work beyond curriculum development and dissemination. For example, establishing certification standards and creating new programs usually must pass through a complex and sometimes bureaucratic administrative process before adoption. But collaboration among professional organizations in education and computing, colleges and universities, state education departments and teachers can help facilitate progress. Consequently, a coordinating entity that supports and sustains the long-term interests of K-12 computer science education must emerge. Perhaps the establishment of a "National Council for Teachers of Computer Science" and/or promoting a more active role for K-12 teachers in the activities of ACM SIGCSE would be helpful in this regard.

5. Conclusions

Computer science is a mainstream discipline that can no longer be ignored by public schools in the 21st century. This model curriculum provides a basis by which states, schools of education, and individual school districts can begin to implement a coherent computer science curriculum that is available to all students.

Much work needs to be done to translate this model into teaching and laboratory materials that are pedagogically viable and widely accessible. We hope that corporations, foundations, and other external sources will support this work by providing appropriate incentives that will enable such a curriculum development effort to succeed.

References

[ACM/IEEE01] ACM/IEEE-CS Joint Curriculum Task Force. *Computing Curricula 2001: Computer Science Volume*. December 2001. http://www.acm.org/sigcse/cc2001/

[ACM93] Task Force of the Pre College Committee of the Education Board of the ACM. ACM model high school computer science curriculum. *Communications of the ACM*, May 1993.

[AP02] *AP Course Description: Computer Science*. May 2002. http://www.collegeboard.com/ap/students/compsci/index.html

[Deek99] Deek, F. and H. Kimmell. Status of computer science education in secondary schools. *Computer Science*

Education 9,2, August 1999.
[Gal99] Gal-Ezer, J. and D. Harel. Curriculum for a high school computer science curriculum. *Computer Science Education* 9, 2, August 1999.
[ISTE02] International Society for Technology in Education (ISTE), *National Educational Technology Standards for Teachers,* June 2002. www.iste.org
[NAS99] National Research Council Committee on Information Technology Literacy, *Being Fluent with Information Technology*, National Academy Press, Washington, DC, May 1999. http://www.nap.edu/catalog/6482.html
[NCATE] Program for Initial Preparation of Teachers of: Educational Computing and Technological Literacy, and Secondary Computer Science Education. http://www.ncate.org/standard/programstds.htm
[Papert] Papert, Seymour. *Mindstorms: Children, Computers, and Powerful Ideas* (1980), and much other information, can be found at http://el.media.mit.edu/logo-foundation/products/books.html#learn
[Stephenson01] Chris Stephenson, summary of the Ontario Curriculum, e-mail correspondence, February 15, 2002.
[Taulbee2002] 2000-2001 Taulbee Survey, *Computing Research News* (March 2002) 4-11. http://www.cra.org/CRN/articles/march02/bryant.vardi.html

Acknowledgments

The design of this curriculum model has been developed with feedback and advice from many persons and groups. We would like to thank the following persons for their valuable contributions and support for the development of this model.

…