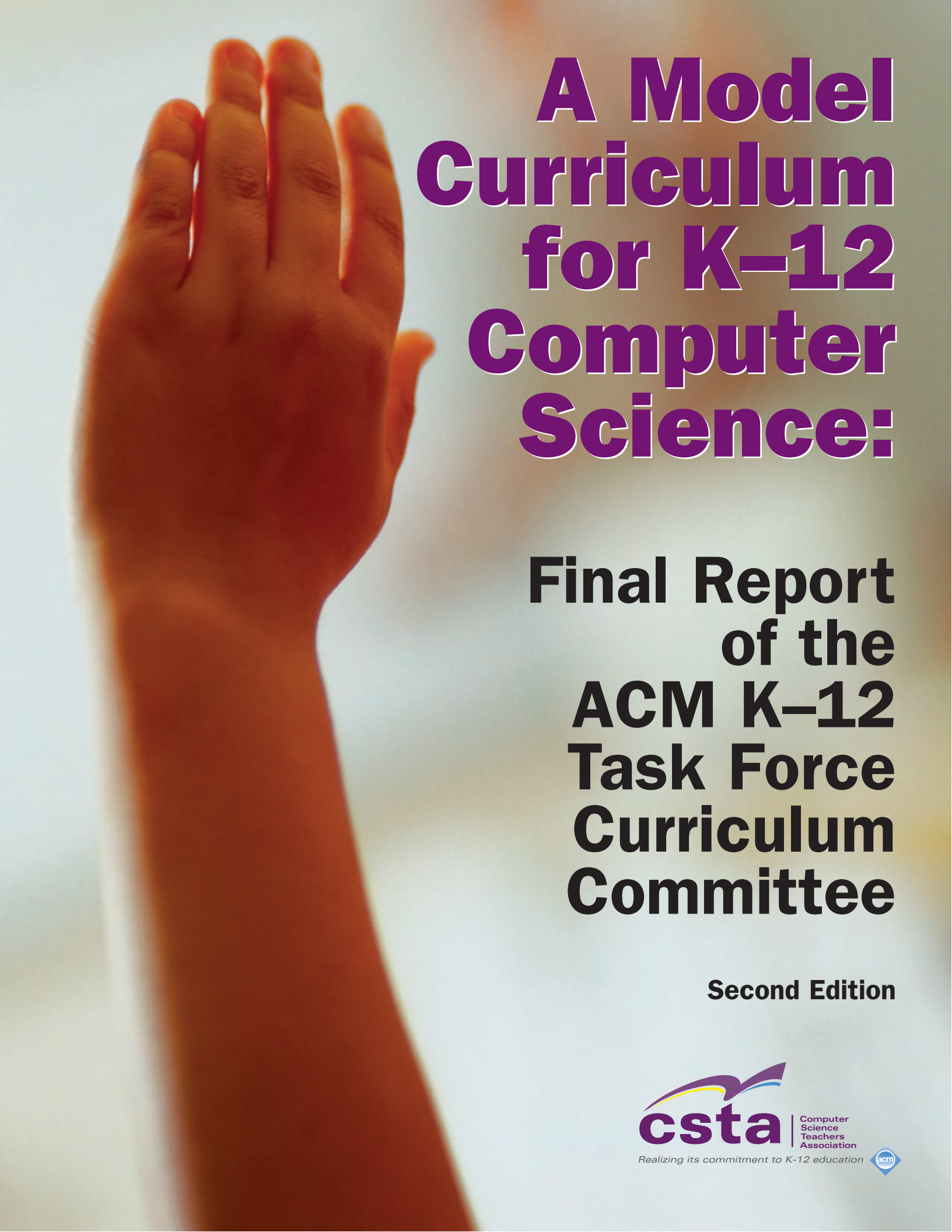

A Model Curriculum for K–12 Computer Science:

Final Report of the ACM K–12 Task Force Curriculum Committee

Second Edition

csta Computer Science Teachers Association

Realizing its commitment to K-12 education

acm

A Model Curriculum for K–12 Computer Science

Final Report of the ACM K–12 Task Force Curriculum Committee October 2003

Allen Tucker
Bowdoin College
Chair
ACM K-12 Task Force Curriculum Committee

Committee Members

Fadi Deek
New Jersey Institute of Technology

Jill Jones
Carl Hayden High School

Dennis McCowan
Weston Public Schools

Chris Stephenson
Executive Director
CSTA

Anita Verno
Bergen Community College

csta

Computer Science Teachers Association
Association for Computing Machinery
2 Penn Plaza, Suite 701
New York, New York 10121-0071

ACM ISBN: # 59593-596-7
ACM Order Number: # 104063
Cost: $15.00

Additional copies may be ordered prepaid from:

ACM Order Department
General Post Office
P.O. Box 30777
New York, NY 10087-0777

Phone: 1-800-342-6626
(U.S.A. and Canada)
+1-212-626-0500
(All other countries)
Fax: +1-212-944-1318
E-mail: acmhelp@acm.org

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 and groups for their valuable contributions and support for the development of this model: Bob Aiken, Moti Ben Ari, Tim Bell, Suzanne Buchelle, Craig Collins, Nancy Head, Peter Henderson, Howard Kimmel, Joe Kmoch, Rich Lamb, NJECC Teachers, Oregon CSTA Teachers, Nick Ourusoff, Seymour Papert, Steve Seidman, Ron Tenison.

We would also like to thank Anne Condon, Dan Frost, Mark Guzdial, Klaus Sutner, and Laurie Williams for creating a new foreword to set this document in the context of important considerations relating to why and how computer science should be taught.

Foreword to the ACM Model Curriculum

Anne Condon, University of British Columbia
Dan Frost, University of California, Irvine
Mark Guzdial, Georgia Institute of Technology
Klaus Sutner, Carnegie Mellon University
Laurie Williams, North Carolina State University

Since its release three years ago, the ACM Model Curriculum for K-12 Computer Science has made a significant contribution to computer science education, providing a practical guideline for educators seeking to ensure that students acquire the skills they need to succeed in an increasingly technology-imbued and globally competitive world and informing the national discourse about the nature of computer science education. To celebrate this second edition, we asked five scholars whose thoughts and work are transforming how computer science is taught to help set the context for this publication.

Education is a complicated undertaking, and education in a fast-moving field like computer science is particularly complicated. So many issues vie for our immediate attention that sometimes it is difficult to find the time or energy to think about one more thing. When Allen Tucker and the members of the ACM K-12 Task Force wrote the *ACM Model Curriculum for K-12 Computer Science* in 2003 it was intended to provide a flexible model for teaching computer science in K-12 that would outline the core concepts and provide appropriate scaffolding for each stage in the learning process.

The Model Curriculum has been widely read and is influencing the computer science curriculum in many U.S. states and Canadian provinces. We have since realized, however, that there are other questions that need to be answered and stories that need to be told. The many misconceptions about computer science, its nature, and the opportunities it provides, motivate us to share our insights about why computer science is one of the most important and relevant academic disciplines.

Computer science has an immense impact on modern life. The job prospects are excellent and the field is rigorous, intellectually vibrant, and multi-faceted. Yet, computer science is in perpetual danger of disappearing from schools. The authors of this section have a wide variety of experiences teaching computer science in the spirit of the *Model Curriculum*. We hope that in sharing our experiences with you, we can help you to understand and advocate for computer science as an essential component of a well-rounded education and a key factor in ensuring that our students have the skills needed, not just to survive, but to thrive in this increasingly technological and global economy.

It is not an exaggeration to say that our lives depend upon computer systems and the people who maintain them to keep us safe on the road and in air, help physicians diagnose and treat health care problems, and play a critical role in the design of new drug therapies. A fundamental understanding of computer science enables students to be not just educated users of technology,

but the innovators capable of using computers to improve the quality of life for everyone.

We have observed that children of all ages love computers. When given the opportunity, even young students enjoy the sense of mastery and magic that programming provides. Older students are drawn to the combination of art, narrative, design, programming, and sheer fun that comes from creating their own virtual worlds. Blending computer science with other interests also provides rich opportunities for learning. Students with an interest in music, for example, can learn about digital music and audio, a field that integrates electronics; several kinds of math; music theory; computer programming; and a keen ear for what sounds beautiful, harmonious, or just plain interesting.

We understand that many obstacles lie in the way of the ideal of a K-12 computer science education for all students. How will room be found in the jam-packed curriculum? How will qualified teachers be recruited, trained, and credentialed? In the world of standards-centric evaluation of schools, should computer science support existing standards, or should new ones be designed for computer science? These and other questions and challenges are significant, but so are the benefits—to students and to society—of computer science becoming as much a part of a high-quality education as biology or physics. The following sections explore some of those benefits.

Computer Science is Important Intellectually

The invention of the computer in the 20th century is a "once in a millennium" event, comparable in importance to the development of writing or the printing press. Computers are fundamentally different from other technological inventions in the past in that they directly augment human thought, rather than, say, the functions of our muscles or our senses. Computers have already had enormous impact on the way we live, think, and act. Yet it is hard to overestimate their importance in the future. In fact, many believe that the true computer revolution will not happen until everyone can understand the technology well enough to use it in truly innovative ways.

So why is it important to study computer science? We live in a digitized, computerized, programmable world, and to make sense of it, we need computer science. An engineer using a computer to design a bridge must understand how the maximum capacity estimates were computed and how reliable they are. An educated citizen using a voting machine or bidding in an eBay auction should have a basic understanding of the underlying algorithms of such conveniences, as well as the security and privacy issues that arise when information is transmitted and stored digitally.

Computer science students learn logical reasoning, algorithmic thinking, design and structured problem solving—all concepts and skills that are valuable well beyond the computer science classroom. Students gain awareness of the resources required to implement and deploy a solution and how to deal with real-world constraints. These skills are applicable in many contexts, from science and engineering to the humanities and business, and have already led to deeper understanding in many areas. Computer simulations are essential to the discovery and understanding of the fundamental rules that govern a wide variety of systems from how ants gather food to how

stock markets behave. Computer science is also one of the leading disciplines helping us understand how the human mind works, one of the great intellectual questions of all time. There is much exciting work that lies ahead of us.

Computer Science Leads to Multiple Career Paths

The vast majority of careers in the 21st century will require an understanding of computer science. Many jobs that today's students will have in 10 to 20 years haven't been invented yet. Professionals in every discipline—from art and entertainment, to communications and health care, to factory workers, small business owners, and retail store staff—need to understand computing to be globally competitive in their fields. Thomas Friedman, in his best-selling book *The World is Flat*, (2006) argues that our economy most needs "Versatilists," people who have expertise in some domain and in technology. Computer science is the glue that makes it possible for these Versatilists to work together.

There is an unmistakable link between success, innovation, and computer science. Movies like *The Incredibles* and *Lord of the Rings* required the development of new computing techniques. Progress on understanding the genetics of disease or of creating an AIDS vaccine requires professionals to think in terms of computer science—because the problems are unsolvable without it. Those who understand the technology can make the new movies and invent the new techniques, and they are the professionals who will go beyond simply using what others have invented.

Studying computer science will prepare a student to become a professional software developer or to pursue a career in one of many related fields. Despite the depressing reports in the media, the reality is that professionals with computer science training have never been more in demand in the U.S. than they are today. Network managers need computer science expertise to install new kinds of routers. So do database designers who help people represent their data in a form that the computer can manipulate. Professional computer scientists rarely spend their days writing program code. More often they are working with experts in many fields, designing and building computer systems for every aspect of our society.

Computer Science Teaches Problem Solving

Artists, philosophers, designers, and scientists in all disciplines are united in the intensely creative activity of problem solving. Every painting by Picasso is an attempt to solve the problem of capturing an active, three-dimensional world on a flat canvas. Every TV commercial during the Super Bowl is an attempt to solve the problem of how to entice people to want, and then purchase, a product. And every well-designed scientific experiment provides data to support or refute a theory.

Computer science teaches students to think about the problem-solving process itself. In computer science, the first step in solving a problem is always to state it clearly and unambiguously. Often a computer scientist works closely with business people, scientists, and other experts to understand the issues, and to define the problem so explicitly that it can be represented in a computer. This co-operative process requires people with different expertise and perspectives to work together to clarify the problems while considering

each other's priorities and constraints. A computer expert helping to design a new computer system for a medical office, for example, has to take into account the current workflow, patient privacy concerns, training needs for new staff, current and upcoming technology, and of course, the budget

Once the problem is well defined, a solution must be created. Computer hardware and peripheral devices must be selected or built. Computer programs must be designed, written, and tested. Existing software systems and packages may be modified and integrated into the final system. In all phases, the computer scientist thinks about resources of computer time and space. Building a system is a creative process that makes our lives better! The process also requires scientific thinking. With each fix of a bug or addition of a new feature, there's a hypothesis that the problem has been solved. Data is collected, results are analyzed, and if the hypothesis is untrue (alas, often!), the cycle repeats.

A computer scientist is concerned with the robustness, the user-friendliness, the maintainability, and above all the correctness of computer solutions to business, scientific, and engineering problems. These issues often require intense analysis and creativity. How will the system respond if the power goes out, or two nurses try to access the same patient record simultaneously, or the insurance company's system is changed, or someone enters unexpected data into the system? Cooperation is again the key. The users and clients have to think about how the system will be used in day-to-day life and anticipate use in the future. Computer specialists draw on their training and experience to avoid problems and to create the best possible solutions.

Computer Science Supports and Links to Other Sciences

Progress in science has always been linked with progress in technology and vice versa. For example, bacteria were first discovered not by a biologist but rather by a Dutch merchant who refined the art of making microscope lenses (and enjoyed peering at plaque he scraped off his unbrushed teeth). Nowadays, it's typical for computer scientists to work in other scientific disciplines. To solve the big scientific problems of the 21st century, such as grappling with new diseases and climate change, we will need people with diverse skills, abilities, and perspectives. And although it may seem surprising, computer science can also help us learn what it really means to be human.

The sequencing of the human genome in 2001 was a landmark achievement of molecular biology, which would not have been possible without computer scientists. After short DNA fragments of the genome were sequenced in biology labs, computers were used to figure out how to piece the fragments together. This knowledge is paving the way for better computational methods of detecting and curing diseases, such as cancer, because we understand the genetic mutations involved.

It doesn't take a neuroscientist to appreciate the fact that the human brain is amazing. We know, for example, that an infant can effortlessly recognize a familiar face from many different viewpoints, and yet, we have a very poor understanding of the computational mechanisms that the brain uses to solve such tasks. Inferring meaning from images is a computational task, and computer scientists and neuroscientists are working together to figure out how to build computers that

can process images and, ultimately, how we can better understand intelligence itself.

The use of modeling and simulation, visualization, and management of massive data sets has created a new field—computational science. This field integrates many aspects of computer science such as the design of algorithms and graphics.

In science classes, students use sophisticated simulation software to make molecules and geological processes come to life. Writing computer programs that model behavior allows scientists to generate results and test theories that are impossible in the physical world. Advances in weather prediction, for example, are largely due to better computer modeling and simulation. Computational methods have also transformed fields such as statistics and mathematics.

Scientists who can understand and contribute to technological innovation have a huge advantage. Good training for future scientists must therefore include a solid basis in computer science.

Computer Science Can Engage All Students

Computer science applies to virtually every aspect of life, so computer science can be explicitly tied to the myriad of student interests. Students may be fascinated with specific technologies such as cell phones or have an innate passion for visual design, digital entertainment, or helping society. K-12 computer science teaching should nurture students' interests, passions, and sense of engagement with the world around them and offer opportunities for them to find purpose and meaning in their lives.

Pedagogically, computer programming has the same relation to studying computer science as playing an instrument does to studying music or painting does to studying art. In each case, even a small amount of hands-on experience adds immensely to life-long appreciation and understanding, even if the student does not continue programming, playing, or painting as an adult. Although becoming an expert programmer, a violinist, or an oil painter demands much time and talent, we still want to expose every student to the joys of being creative. The goal for teaching computer science should be to get as many students as possible enthusiastically engaged with every assignment. Instead of writing the same old mortgage calculation program, have students design and write programs that control their cell phones or robots, create physics and biology simulations, or compose music. Students will want to learn to use conditionals, loops, and parameters and other fundamental concepts just to make these exciting things happen.

In a fast-paced field such as computer science, we are all challenged to keep up with our peers and our students. Technology changes rapidly, and students are more likely than teachers to be familiar with the latest incarnations. No teacher should ever be ashamed of learning from her or his students. Real learning involves everyone in the room living with a sense of wonder and anticipation.

We know that teaching computer science involves some unique challenges and that none of us has all of the answers. Here are just a few suggestions that we have found helpful in our attempts to better interest, engage, and motivate our university students. Not all of them will be completely applicable to the high school classroom, but we believe that they contain useful and varied suggestions that may inspire both students and teachers. (Please see the CSTA web repository at *http://csta.acm.org/Resources/sub/WebRepository.html* for a comprehensive collection of resources for teach-

ing and learning computer science that correspond directly to the recommendations of the ACM Model Curriculum for K-12 Computer Science.)

Computer Science and Digital Media

Mark Guzdial, Georgia Institute of Technology

Manipulating and creating digital media is a context that engages students and easily integrates with computer science learning goals. Instead of iterating over an array to compute an average, students might write a program to iterate over an array of pixels to compute a negative image or a grayscale image. Students can learn that combining two arrays is the technique used to splice and mix digital sounds. Linked lists are much more exciting when the nodes contain music or pictures, so that traversing the list plays a song or generates a frame of an animation. Several schools are now using this technique and are reporting increased engagement and less attrition of students in computer science classes. The key strategy used here is finding a context that meshes with students' interests and motivations where even simple programs result in tangible and fun results. Similar contexts are robotics and story-telling with digital media.

Pair Programming

Laurie Williams, North Carolina State University

Pair programming refers to the practice whereby two programmers work together at one computer, collaborating on the same design, algorithm, code, or test. The pair is made up of a driver, who actively types at the computer or records a design; and a navigator, who watches the work of the driver and attentively identifies problems, asks clarifying questions, and makes suggestions. Both are also continuous brainstorming partners. Throughout the world, many universities are using pair programming in their computer science classes—and a number of high schools have begun using the practice as well. Generally, students much prefer to collaborate than to work alone. Between the two students, they can generally figure out most problems and can avoid pesky syntax and semantic errors that can cost many hours to debug. Perhaps during those multi-hour debugging sessions (that are greatly reduced with pair programming) some students vow to never take another computer science course! Educators who have used pair programming report higher retention and student success in the courses, higher enrollment in future computer science courses, happier and less frustrated students, and equal or higher grades on exams and projects.

Computational Thinking

Klaus Sutner, Carnegie Mellon University

How does one prevent a computer from creating many thousands of e-mail accounts that can be used to send spam to millions of people? How can one design an electronic auction system that fairly represents the interests of all parties involved? How can one accurately simulate a system consisting of millions of objects evolving over billions of steps? To deal with these problems, and many more similar ones, requires a type of thinking characteristic of computer science: computational thinking. Computational thinking involves a clear focus on tangible problems; a large collection of proven techniques such as abstraction, decomposition, iteration, and recursion; an understanding of the capabilities of humans and machines alike; and a keen awareness of the cost of it all. Emphasis on computational thinking rather than

just programming has greatly improved introductory courses and is starting to become a motivating principle in other parts of our curriculum. Carnegie Mellon University offers a few newly designed courses as part of its undergraduate curriculum. The first, Principles of Computation, is targeted at students with no prior background who are interested in key principles rather than programming. The second, Computational Discrete Mathematics, presents topics in discrete mathematics in an experimental and experiential way by strongly emphasizing the computational aspects of the material.

Computers and the Visual Arts:

Anne Condon, University of British Columbia

In our Computers and Visual Arts module, we describe how computer technology and graphic arts have developed in parallel. We use standard computer paint programs to illustrate how images can be represented digitally. We describe the principles underlying different file types (such as jpg files). Using a standard programming language, we then introduce algorithmic ways of representing geometric images, and introduce loops (as well as if statements) as a way to create geometric patterns. Finally, we introduce, using purely visual methods, generative systems, which can be used to represent complex images, such as plants, and convey the concept of recursion to students.

Computers and Biology:

Anne Condon, University of British Columbia

In our Computers and Biology module, we focus on molecular biology, and the sequencing of the human genome. Through a short introduction to DNA, we explain how our genetic heritage is represented digitally in our genome. Then we introduce the fragment assembly problem, which was a central computational task in sequencing the human genome. This problem is a nice vehicle for introducing the fact that some computational problems seem to have no efficient solution—a deep insight of computer science. One current disadvantage of this approach is the lack of good resource materials, but this is changing rapidly. Lecture slides and lecture notes are available by request from Anne Condon (*condon@cs.ubc.ca*).

contents

A Model Curriculum for K–12 Computer Science

Final Report of the ACM K–12 Task Force Curriculum Committee October 2003

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—Computer Science Teacher's Association
Anita Verno—Bergen Community College

Executive Summary

This report proposes a model curriculum that can be used to integrate computer science fluency and competency throughout primary and secondary schools, both in the United States and throughout the world. It is written in response to the pressing need to provide academic coherence to the rapid growth of computing and technology in the modern world, alongside the need for an educated public that can utilize that technology most effectively to the benefit of humankind.

Computer science is an established discipline at the collegiate and post-graduate levels. Oddly, the integration of computer science concepts into the K–12 curriculum has not kept pace in the United States. As a result, the general public is not as well educated about computer science as it should be, and a serious shortage of information technologists at all levels exists and may continue into the foreseeable future. This curriculum model aims to help address these problems. It provides a framework within which state departments of education and school districts can revise their curricula to better address the need to educate young people in this important subject area, and thus better prepare them for effective citizenship in the 21st century.

This curriculum model provides a four-level framework for computer science, and contains roughly the equivalent of four half-year courses (many of these can be taught as modules, integrated among existing science and mathematics curriculum units). The first two levels suggest subject matter that ought to be mastered by all students, while the second two suggest topics that can be elected by students with special interest in computer science, whether they are college-bound or not. The Appendix to this report provides "proof of concept" by outlining existing courses and modules that are now being taught in different school districts at each of the four levels.

These recommendations are not made in a vacuum. We understand the serious constraints under which school districts are operating and the up-hill battle that computer science faces in the light of other priorities, as well as time and budget constraints. Thus, we conclude this report with a series of recommendations that are intended to provide support for a long-term evolution of computer science in K–12 schools. Many follow-up efforts will be needed to sustain the momentum we hope this report will generate. Teacher training, curriculum innovation, in-class testing, textbook and Web site development, and dissemination are but a few of the challenges.

We hope this report will serve as a catalyst for widespread discussions and the initiation of many pilot projects that can take the evolution of K–12 computer science to the next level. We invite you to read the entire report, and then to take part in this discussion in a way that mutually benefits both you and the K–12 education community. More information about ongoing activities that are related to this effort can be found at: *http://csta.acm.org/*.

A Model Curriculum for K–12 Computer Science

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 bound or workplace bound.

Much evidence (National Research Council, 1999) confirms an urgent need to improve the level of public understanding of computer science as an academic and professional field, including its distinctions from management information systems (MIS), information technology (IT), mathematics, and the other sciences. Elementary and secondary schools have a unique opportunity and responsibility to address this need. That is, to function in society, the average citizen in the 21st century must understand at least the principles of computer science. A broad commitment to K–12 computer science education not only will create such broad public understanding but also will help to address the worldwide shortage of computer specialists. The creation of a viable model for a computer science curriculum and its implementation at the K–12 level is a necessary first step toward reaching these goals.

This report addresses the entire K–12 range. Its recommendations are therefore not limited 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 (AP, 2002) and the National Educational Technology Standards (NETS) curriculum (ISTE, 2002).

At this time, the development of state-level curriculum standards for computer science in the United States 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 standards) or as a collection of topics integrated with other science curricula (e.g., Maine's "Learning Results" (State of Maine, 1997). 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.

All drafts of this report have been informed by feedback from many sources; we hope that this final draft will receive widespread dissemination and continued scrutiny from everyone who has interests or experience in K–12 computer science education. To that end, this report is published on the Computer Science Teachers Association (CSTA) Web site (*http://csta.acm.org/Curriculum/sub/ACMK12CSModel.html*) as well as in hardcopy. Feedback has been actively sought from the following professional organizations:

- Academy of Information Technology/National Academy Foundation (AOIT/NAT)
- Association for Computing Machinery (ACM) Special Interest Group for Computer Science Education (SIGCSE)
- ACM Education Board
- Association for Supervision and Curriculum Development (ASCD) Curriculum Directors in school districts
- Institute of Electrical and Electronics Engineers (IEEE) Computer Society Educational Activities Board
- International Society for Technology in Education (ISTE) Special Interest Group for Computer Science (SIGCS)
- National Association of Secondary School Principals (NASSP)
- National Education Association (NEA)
- National School Boards Association (NSBA)

In addition, presentations of this report at ISTE's National Educational Computing Conference (NECC) and ACM's SIGCSE Symposia have provided valuable opportunities for dissemination and feedback.

We recognize that many of the recommendations in this report are so ambitious as to be beyond the reach of most school districts at the present time. However, rather than do nothing, we offer this work as a comprehensive and coherent model, one that can be used as the basis for beginning a dialogue—an ideal toward which many districts can evolve over time. This report thus provides a catalyst for a long-term process—it defines the "what" from which the "how" can follow during the next several years.

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 algorithmic processes[1], including their principles, their hardware and software designs, their applications, and their impact on society.

In our view, this definition requires that K–12 computer science curricula have the following kinds of elements: programming, hardware design, networks, graphics, databases and information retrieval, computer security, software design, programming languages, logic, programming paradigms, translation between levels of abstraction, artificial intelligence, the limits of computation (what computers *can't* do), applications in information technology and information systems, and social issues (Internet security, privacy, intellectual property, etc.).

Typically, K–12 science and mathematics curricula do not cover any significant amount of these topics, nor do they identify what they do cover as elements of computer science. However, some of the emerging K–12 information technology curricula are addressing some of them, especially the applications and social impact of computers. However, there is strong evidence (National Research Council, 1999) that a basic understanding of all these topics is now an essential component for preparing high school graduates for life in the 21st century.

The goals of a K–12 computer science curriculum are to:

1. introduce the fundamental concepts of computer science to all students, beginning at the elementary school level.
2. present computer science at the secondary school level in a way that would be both accessible and worthy of a curriculum credit (e.g., math or science).
3. 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.
4. increase the knowledge of computer science for all students, especially those who are members of underrepresented groups.

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, and to summarize the nature of CS at the college and university level.

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

[1] An algorithm is a precise, step-by-step description of a solution to a problem. Programming is used to implement algorithms on computers. 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.

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 (National Research Council, 1999) 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 (National Research Council, 1999) 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 are not 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 has scientific and mathematical, as well as practical, dimensions. Some of the practical dimensions of computer science are shared with IT, such as working with text, graphics, sound, and video. But while IT concentrates on learning how to use and apply software as a tool, computer science is concerned with learning how these tools are designed. This latter concern exposes students to the scientific and mathematical theory that underlies the practice of computing. Therefore, any comprehensive K-12 computer science curriculum will necessarily have topics that are distinct from those that normally appear in an IT curriculum.

The idea of IT fluency (National Research Council, 1999) 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*.

Concepts are the 10 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.

Capabilities are the 10 fundamental abilities for using IT to solve a problem:

> Engage in sustained reasoning, manage complexity, test a solution, manage faulty systems and software, organize and navigate information structures and evaluate information, collaborate, communicate to other audiences, expect the unexpected, anticipate changing technologies, and think abstractly about IT.

Skills are the 10 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 National Research Council, 1999) 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 United States 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 45,000 baccalaureate and 850 PhD degrees each year (Taulbee, 2002).

The current model for college computer science major programs was published in 2001 (ACM/IEEE, 2001). 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* (HCI) (8): principles of HCI, building a graphical user interface (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, object-oriented (OO) programming, functional programming, translation
- *Social and Professional Issues* (16): ethical responsibilities, risks and liabilities, intellectual property, privacy, civil liberties, crime, economics, impact of the Internet
- *Software Engineering* (31): metrics, requirements, specifications, design, validation, tools, management

Undergraduate computer science programs also provide students with regular 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 are expected to 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, 2001):

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.

The presence of a K–12 computer science program should allow pre-college students to begin developing these capabilities and skills.

2.3 The Current Status of K–12 Computer Science

Computer science has never been widely taught at the K–12 level in the United States. To help address this problem, the ACM Model High School Curriculum (ACM, 1993) 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. 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, spreadsheets, word processing, and desktop publishing. Its electives included topics such as AI (expert systems, games, robotics), 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 (Deek, 1999), 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 grades 9, 10, and 12, in a way that complements the AP computer science curriculum (offered in the grade 11). The grade 9 course provides

an introduction to programming and problem solving, the Internet, information, communication, hardware, social impact and ethics; the grade 10 course emphasizes programming and applications. At grade 12, a "topics" course provides an opportunity to offer interesting subjects like robotics, simulations, and animation.

In spite of these efforts, a survey conducted in 2002 (*http://csta.acm.org/Research/sub/CSTAResearch-2.html*) confirms that neither the 1993 ACM model nor any other model has achieved widespread recognition or implementation in the United States. Seventy respondents, representing 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 grades 9 and 10). So, even for states that offer any computer science courses, there is much divergence in the number and content of these courses. Where they are offered, computer science courses also 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 (Deek, 1999).

The development of K–12 computer science is making more headway internationally than in the United States.

In Israel, a secondary school computer science curriculum (Gal-Ezer & Harel, 1999) was approved by the Ministry of Higher Education and implemented in 1998. It blends conceptual and applied topics, and is offered in grades 10, 11, and 12. All students in grade 10 are required to take a half-year course in the foundations of computer science. This is followed by 1½ or 2½ years of electives taught at grades 11 and 12. These electives have a particularly heavy emphasis on the foundations of algorithms.

In Canada, a comprehensive curriculum was recently implemented for all secondary schools in Ontario (Stephenson, 2002). 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 grade 9, a full-year "integrated technologies" course is available to all students. This is followed by three parallel three-year tracks—one in computer and information science and two in computer engineering.

In many other parts of the world, including Europe, Russia, Asia, South Africa, New Zealand, and Australia, computer science is being established in the K–12 curriculum. Thus, we feel a certain sense of urgency about the establishment of computer science in the United States—this nation's educated workforce should remain competitive with that of other nations in its level of understanding about computer science in the modern world.

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 focuses on fundamental concepts and 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. One simple example is the use of logic for understanding the semantics of English in a language arts class. There are many others.
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 has four different levels, whose goals and content are introduced below.

Level I (recommended for grades K–8) should provide elementary school students with foundational concepts in computer science by integrating basic skills in technology with simple ideas about algorithmic thinking. This can be best accomplished by adding short modules to existing science, mathematics, and social studies units. A combination of the NETS (ISTE, 2002) standards and an introduction to algorithmic thinking (as offered, for instance, by Logo (Papert, 1980) or other hands-on experiences (Bell, 2002) 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 best be offered as a one-year course accessible to all

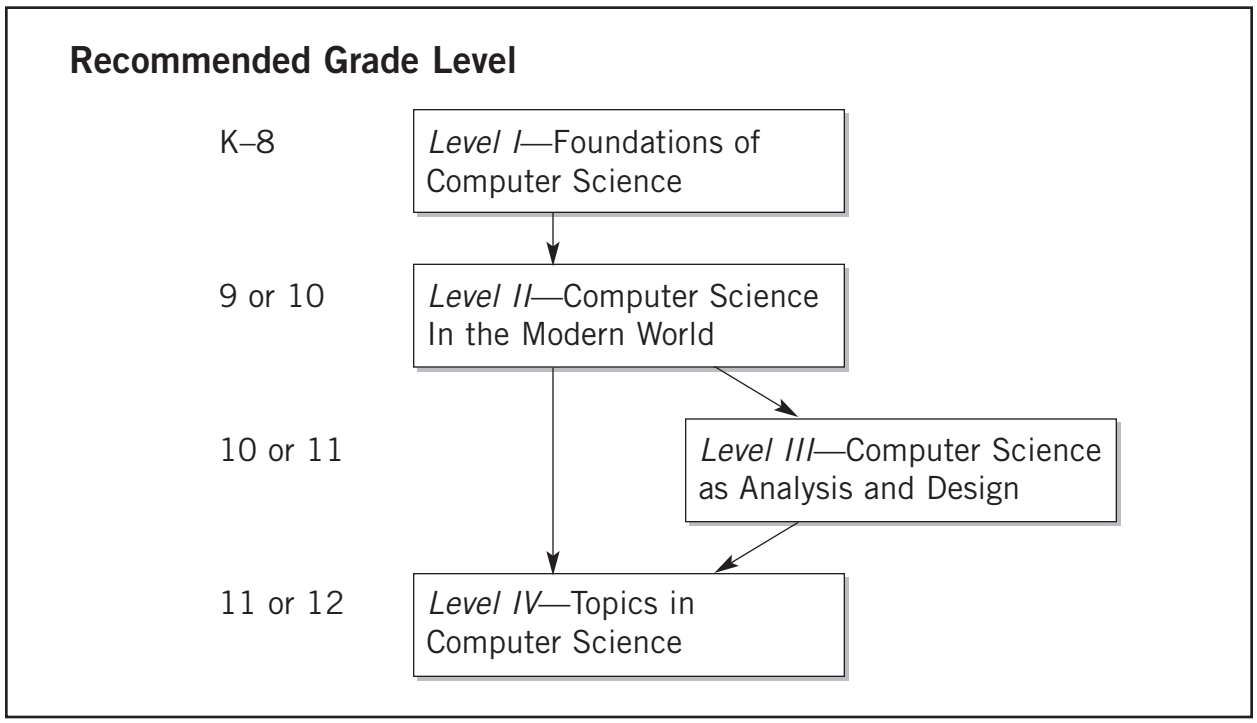

Figure 1. Structure of a K–12 Computer Science Curriculum

students, whether they are college-bound or workplace-bound. Since, for most students, this *Level II* course will be their last encounter with computer science, it should be considered essential preparation for the modern world.

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 a curriculum credit (e.g. math or science). 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 (AP, 2002) 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 II course as a prerequisite, and some will require the Level III course as well.

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

3.1 Level I—Foundations of Computer Science

Because the foundations of computer science have 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.

3.1.a. Topics and Goals

The National Educational Technology Standards (NETS) (ISTE, 2002) provide an excellent starting place for defining requirements for elementary student preparedness in computer science.[3]

[2] Too frequently, new and complex expectations are put on 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 given to preparing teachers to use the technology effectively and in support of their own teaching and learning goals.

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 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.

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 method for solving a problem in a step-by-step manner. So children learn about algorithmic problem solving whenever they discover a collection of steps that can be carried out to accomplish a task. These steps should 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.

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

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 teachers 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 encourage students to engage 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, 1980) provide many more examples of how K–8 students can be engaged in algorithmic thinking. Additional examples of computer science topics appropriate for the K–8 level are included in the next section.

3.1.b. Grade-Level Breakdowns

To ensure that students achieve these goals, we paraphrase here the NETS model (ISTE, 2002), which identifies different sets of outcomes for three different groups of students: grades K–2, grades 3–5, and grades 6–8. We have augmented that model by adding outcomes that engage students with algorithmic thinking and other foundational elements of computer science.

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.
11. Understand how 0s and 1s can be used to represent information, such as digital images and numbers.
12. Understand how to arrange (sort) information into useful order, such as a telephone directory, without using a computer (see Appendix for examples).

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 occur in electronic information sources.
11. Develop a simple understanding of an algorithm, such as text compression, search, or network routing, using computer-free exercises (see Appendix for examples).

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 effects 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.
11. Understand the graph as a tool for representing problem states and solutions to complex problems (see Appendix for examples).
12. Understand the fundamental ideas of logic and its usefulness for solving real-world problems (see Appendix for examples).

3.2 Level II—Computer Science in the Modern World

This is a one-year course (or the 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.

3.2.a. Topics and Goals

A major outcome of this course (or its equivalent) is to provide students with general knowledge about computer hardware, software, languages, networks, and their impact in the modern world.[4] That is, since most students at Level II will eventually encounter computers and networks as users, the overarching aim

here is to prepare students to master computer science concepts from the user's point of view rather than from the designer's. 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. Teaching students about various input devices currently in use should help demystify the general idea of input, and prepare students to be comfortable using devices with which they are not yet familiar.

Students should gain a conceptual understanding of the following topics 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, routing protocols for connection/communication, spoolers and queues, shared resources, and fault-tolerance).
4. Organization of Internet elements, Web page design (forms, text, graphics, client- and server-side scripts), 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), and what distinguishes humans from machines.
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), and the positive and negative impact of technology on human culture.
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).

3.2.b. 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 algorithmic design and programming component of the course should include the following:

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

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

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

- The use of hypertext links to load new pages or activate processes
- Storing, compressing, encrypting, and retrieving image, video, and sound data
- User interface design
- Tools for expressing design (storyboard, site map)

3.2.c. 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 toward 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 2.1). 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 uses in other academic subjects, this course is a study of computer science as an academic subject *per se*.

Several example activities that can be used to teach this course are shown in the Appendix.

3.3 Level III—Computer Science as Analysis and Design

This is a one-year course (or the equivalent) that should earn curriculum credit (e.g., science or math). 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 to integrate programming and design with complex "real world" projects.

3.3.a. 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. Topics in discrete mathematics: logic, functions, sets, and their relation to computer science
4. Design for usability: Web page design, interactive games, documentation
5. Fundamentals of hardware design
6. Levels of language, software, and translation: characteristics of compilers, operating systems, and networks
7. The limits of computing: what is a computationally "hard" problem? (e.g., ocean modeling, air traffic control, gene mapping) and what kinds of problems are computationally unsolvable (e.g., the halting problem)

8. Principles of software engineering: software projects, teams, the software life cycle
9. Social issues: software as intellectual property, professional practice
10. Careers in computing: computer scientist, computer engineer, software engineer, information technologist

3.3.b. 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

3.3.c. 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 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 section 3.4), 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 an IT curriculum, such as the limits of computing and data structures.

Example activities that have been used in this kind of course are shown in Appendix A.3.

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 II course as a prerequisite, while some may require the Level III course as well. Most important, these courses provide students with an opportunity to explore topics of personal interest in greater depth, and thus prepare for the workplace or for further study at the post-secondary level.

These electives include, but are not necessarily limited to:

- Advanced Placement (AP) Computer Science
- 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.

3.4.a. AP Computer Science

The AP Computer Science curriculum is well established (AP, 2002), and is offered at many secondary schools for students planning to continue their education in a two- or four-year college or university, possibly in computer science, business, or a related field.

Students taking an AP course should have completed Levels I and II. 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 III overlap somewhat with the AP course, so some of the AP course can serve as a review if students have had the Level III 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 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 and 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 levels increase.

In schools that implement this curriculum recommendation, students will arrive at Level IV 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 trade-offs that result from offering 0, 1, or 2 AP courses in a setting that also offers the Level II and III courses described above.[5] For example, a school that is neither large nor resource-rich may prefer to offer the Level III course alone, and then supplement that course with additional material that will support a smaller group of students preparing for the AP A exam.

Example modules that can be used to teach this course are shown in the Appendix A.4.

3.4.b. Project-Based Courses

This kind of course would be available to all students who have completed the Levels I and II curricula. Some variants of this course would also require completion of Level III (see below). This could be either a half-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.

[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 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.

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 each task is 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 projects that could populate such a course. See the Appendix for more details.

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 I) will help ensure 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 II).

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 I) and multimedia concepts (Level II) 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 I) and an understanding of multimedia concepts (Level II) 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 creating 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 III—limits of computers and design for usability.

EXAMPLE: Design and Development of Web Pages. At Level II, 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 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 III 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) includes:

- The computer and animation (Level II)
- Networking technologies (Level III)
- Programming simulations (e.g., a computer-controlled chemistry experiment) (Level III)
- Object-oriented design and coding (Level IV—AP computer science)
- Effective use of computer applications (Level II)

3.4.c. Courses Leading to 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 two-year college AAS program. Students taking this course should have completed the Level I and Level II 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 Levels I and II.

EXAMPLE: Certified Internet Webmaster (CIW). "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 Levels I and II. 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 infrastruc-

ture 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 *http://www.computer-certification-training.com/index.html*.

Further discussion and examples of these kinds of courses are provided in the Appendix.

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.

For schools to widely implement this model, work is needed in three important areas: *teacher preparation, state-level content standards, and curriculum materials development*. In addition, persons in leadership positions must acknowledge the importance of computer science education for the future of our society. States and accrediting organizations should make this a factor in overall school accreditation. 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 ACM, the IEEE Computer Society, ISTE, institutions of higher education, and national and local teacher organizations. Below is a discussion of the main challenges as we see them.

4.1 Teacher Preparation

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 prospective 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 33 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/IEEE, 2001). 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 and 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

According to the 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 review their licensing standards for professional educators so that they recognize and support computer science as a distinct 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 evolving rapidly.

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 the 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, or English, but they have since self-educated to teach many different types of computing courses, including AP computer science. One way to provide such recognition is for states to develop standard core competencies for computer science teachers and endorse 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.

4.2 State-Level Content Standards

Recently, efforts have increased to develop national and state content standards for computer science. 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.

4.3 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 information 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 could be made up of working groups and 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.

4.4 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 SIGCS, ASCD curriculum directors in school districts, state boards of education, NEA, NASSP, and NSBA.

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—Support the establishment of a "National Computer Science Teachers Association," a new professional society for K–12 computer science teachers, which has recently been proposed by ACM (ACM, 2003). Similarly, ACM SIGCSE and ISTE's NECC should continue to broaden their missions and conferences to better accommodate K–12 computer science teachers. State and regional organizations should provide ongoing support and collaboration for K–12 computer science teachers at the local level.

Culture—Most teachers who now offer computer science in K–12 schools are experiencing a strong sense of isolation and vulnerability. This frustration has many roots, including the glacially slow pace of attitudinal and programmatic change, the battle to obtain adequate computing resources, the lack of acceptance of computer science among math and science colleagues, the absence of state curricular standards, the shortage of opportunities for in-service and pre-service training in computer science, and the unusual vulnerability of computer science faculty and courses to budget cuts during times of fiscal restraint. All of these combine to create an atmosphere in which a culture of computer science among K–12 teachers is almost impossible to create or sustain at a significant level. If computer science is to become a meaningful and effective academic culture within the mainstream of K–12 education, a significant change in all of these inhibiting factors must take place. Until then, teachers will continue to struggle to keep any sort of computer science presence in their school's curriculum, and schools that do not now offer computer science will probably never consider making such a change.

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, new certification standards and 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.

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 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

Proposal to form the National Computer Science Teachers Association (NCSTA), Association for Computing Machinery, 2003 (unpublished).

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

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.

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

Bell, T., I. Witten, and M. Fellows, *Computer Science Unplugged*, June 2002. *http://ww.unplugged.canterbury.ac.nz*

Deek, F. and H. Kimmel. Status of computer science education in secondary schools. *Computer Science Education* 9,2, August 1999.

Friedman, T.L. The World Is Flat: A Brief History of the Twenty-first Century. Farrar, Straus and Giroux, 2006.

Gal-Ezer, J. and D. Harel. Curriculum for a high school computer science curriculum. *Computer Science Education* 9(2), August 1999.

International Society for Technology in Education (ISTE), *National Educational Technology Standards for Teachers*, June 2002. *http://www.iste.org/standards/*

State of Maine Learning Results, *http://www.state.me.us/education/lres/lres.htm*

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*

National Council for Accreditation of Teacher Education. 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, 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*

Stephenson, C. E-mail correspondence with a summary of the Ontario Curriculum. February 15, 2002.

2000–01 Taulbee Survey, *Computing Research News* (March 2002) 4–11. *http://www.cra.org/CRN/articles/march02/bryant.vardi.html*

Appendices

In these appendices, we illustrate the viability of this curriculum model by providing example activities for courses and modules that are now being taught in various schools throughout the world. Many of these are directly adapted from the Ontario Computer Science Curriculum discussed in this report. The layout of these activities (also adapted from the Ontario curriculum) is explained below:

Activity: . . . . . Name of the activity
Time: . . . . . Number of in-class hours to complete the activity
Description: . . . . . Brief description of the subject and goals of the activity.
Level: . . . . . I, II, III, or IV, as defined in this report (Section 3)
Topics: . . . . . The topics at this level that are covered by this activity (see the topic lists in Sections 3.1, 3.2, 3.3, and 3.4 of this report)
Prior Knowledge: . . . . . What students should know before beginning this activity
Planning Notes: . . . . . Suggestions to teachers for preparing this activity
Teaching/Learning Strategies: . . . . . Organization of the in-class presentation and the particular student tasks
Assessment and Evaluation: . . . . . Formative and summative assessments of in-class and laboratory work
Accommodations: . . . . . Additional supporting materials (e.g., scaffolding labs, example programs, challenging problems)
Resources: . . . . . Links to the source of this activity, as well as other related activities

A.1 Sample Activities for Level I: Foundations of Computer Science

Activity: . . . . . Color by Numbers
Time: . . . . . 3 hours
Description: . . . . . The computer stores drawings, photographs, text, and other pictures using only numbers. This activity demonstrates how that is done.
Level: . . . . . I (K–2)
Topics: . . . . . 11—using 0s and 1s to represent information
Prior Knowledge: . . . . . Grade 2 geometry (exploring shapes), counting, graphing
Planning Notes:

- Motivational discussion questions include, "What does a fax machine do?"
- "In what situations would a computer want to store pictures?"
- "How do computers store pictures when they can only use numbers?"

Teaching/Learning Strategies:

- A 5x6 rectangular grid is used as a basis for representing different images (such as letters) by coloring in some of the squares (pixels).
- Coding of the image is done by scanning the sequences of 1s (shaded squares) and 0s in each row of the grid and recording the length of each sequence.

Assessment and Evaluation: . . . . . Worksheet activities.
Accommodations: . . . . . No computers are required; students use two worksheet activities, called "Kid fax" and "Make your own picture"
Resources: . . . . . See *www.unplugged.canterbury.ac.nz* to learn more about this activity.

Activity: Beat the Clock
Time: 4 hours
Description: There is a limit to how fast computers can solve problems. One way to speed them up is to use several computers to solve different parts of a problem. This activity uses sorting networks to do several comparisons at the same time.
Level: I (Grades K–2)
Topics: 12—understanding how to arrange (sort) information into useful order
Prior Knowledge: Grade 2 mathematics; greater than, less than
Planning Notes: Some tasks can be done faster using fewer steps, while others can be done faster using parallel computation. Sorting networks is a good example of the latter.

Teaching/Learning Strategies:

- Six students hold one number each and arrange themselves on the left-hand side of the court. They move forward to the next circle in the network and wait for someone else to arrive.
- The circle is a decision point from which the student with the smaller number goes left and the larger number goes right.
- At the end, the numbers are sorted.

Assessment and Evaluation:

- Students successfully sort the numbers using the given network.
- They also discuss the use of other networks for sorting.

Accommodations: This is an outdoor group activity. Chalk, two sets of six cards with numbers on them, and a stop watch are needed. A master is used to draw a sorting network on the sidewalk.
Resources: See *www.unplugged.canterbury.ac.nz* to learn more about this activity.

Activity: You Can Say that Again
Time: 4 hours
Description: Text can be compressed by taking advantage of patterns in words and linking repeating patterns to each other without rewriting them. For instance, "pitter patter" can be encoded by replacing the last instance of "tter" by a link to the first instance.
Level: I (Grades 3–5)
Topics: 11—develop a simple understanding of an algorithm
Prior Knowledge: English, recognizing patterns in words, copying written text; basic familiarity with computers
Planning Notes: Images and text containing millions of pieces of information are transmitted on the Internet every day. To save time and space, they are compressed into ZIP or GIF format before they are transmitted.

Teaching/Learning Strategies:

- Students successfully encode and decode text, using worksheets.
- They also discuss the kinds of texts and images that compress best/worst using this algorithm.

Assessment and Evaluation:

- Students' completed worksheets are evaluated as ordinary math assignments.

Accommodations: Four different worksheets are used to facilitate this activity; a transparency is used to present the compression algorithm.

Resources: See *www.unplugged.canterbury.ac.nz* to learn more about this activity.

Activity: Battleships

Time: 4 hours

Description: Computers are often required to find information in large collections of data. They need to develop quick and efficient ways of doing this. This activity demonstrates three different search methods—linear search, binary search, and hashing—using numbered cards and the game of battleships as vehicles.

Level: I (Grades 3–5)

Topics: 11—basic understanding of a search algorithm

Prior Knowledge: Mathematics; greater, less, and equal relationships, geometry (coordinates)

Planning Notes: Finding information efficiently—linear search, binary search, hashing

Teaching/Learning Strategies:

- 15 children have cards with different numbers on them, arranged randomly and hidden from one of the children who tries to guess who holds a mystery number. The game is repeated after the 15 numbers are rearranged in order.
- Children are grouped in pairs, and each pair is given two battleship game cards. The game is played using a simple hashing technique to locate the column of a ship on the card.

Assessment and Evaluation:

- Discussions should explore the scores children achieved in each game.
- Discussions should also explore the advantages and disadvantages of each search strategy.

Accommodations: Each child will need a game card (copied from masters).

Resources: See *www.unplugged.canterbury.ac.nz* to learn more about this activity.

Activity: The Orange Game

Time: 4 hours

Description: This activity uses a simple game with oranges to illustrate Internet traffic management (routing) and deadlocks.

Level: I (Grades 3–5)

Topics: 11—simple algorithms for network routing

Prior Knowledge: Math; logic and reasoning

Planning Notes: This is a group activity, requiring five or more children sitting in a circle and having different letters on their shirts. There are two oranges for each child's letter except one, for which there is only one orange.

Teaching/Learning Strategies:

- Every child is given an orange in each hand (except that one child has one orange) randomly.

- Oranges are passed between children until everyone has two oranges with his/her own letter.
- Only an empty hand can receive an orange, and only from an adjacent hand.

Assessment and Evaluation:

- Children should learn that holding onto one's own orange as soon as it is received may prevent the whole group from achieving its goal.

Accommodations: . . . . . A bag of oranges

Resources: . . . . . See *www.unplugged.canterbury.ac.nz* to learn more about this activity.

Activity: . . . . . Ice Cream Stand Problem

Time: . . . . . 4 hours

Description: . . . . . A graph is used to represent the map of a city. An ice cream company wants to build ice cream stands at different intersections, so that it is easy for people to get to them but not too many stands have to be built.

Level: . . . . . I (Grades 6–8)

Topics: . . . . . 11—understand the graph as a tool for representing problem states and solutions

Prior Knowledge: . . . . . Elementary map reading

Planning Notes:

- Pass out copies of a map of a town, where lines represent streets and circles represent intersections.

Teaching/Learning Strategies:

- Children must determine the smallest number of stands to build so that no person has to walk to more than one intersection to buy ice cream.

Assessment and Evaluation:

- Children discuss different strategies for placing the stands, and they evaluate each other's solutions.

Accommodations: . . . . . Copies of different city maps need to be handed out.

Resources: . . . . . See *http://www.c3.lanl.gov/mega-math/menu.html* to learn more about this activity.

Activity: . . . . . A Mystery Play

Time: . . . . . 8 hours

Description: . . . . . Students learn and act out a short mystery play. Other students use logic to solve the mystery.

Level: . . . . . I (Grades 6–8)

Topics: . . . . . 12—understand the fundamental ideas of logic and its use for solving real-world problems

Prior Knowledge: . . . . . Elements of logic (statements, truth and falsity), reading and speaking skills.

Planning Notes:

- Students will need time to learn their parts and rehearse the play in advance.

Teaching/Learning Strategies:

- The play is about 20 minutes long.
- The setting is a classroom, so no special props or scenery are needed.

Assessment and Evaluation:

- Students discuss the mystery and the reasoning they used to solve the mystery.

Accommodations: Students receive copies of the play, which is 3 scenes and 4 pages long.

Resources: See *http://www.c3.lanl.gov/mega-math/menu.html* to learn more about this activity.

A.2 Sample Activities for Level II: Computer Science in the Modern World

Activity: Number Systems

Time: 4 hours

Description: Students develop an understanding of the relationship between the binary number system and computer logic. Also, students learn how to convert Base 10 numbers into binary and vice versa. Character representation of binary codes is explored. Students have the opportunity to experiment in writing their own message and decoding.

Level: II (Grades 9-10)

Topics: 6—the connection between elements of mathematics and computer science, including binary numbers, logic, sets and functions.

Prior Knowledge: Understanding of the decimal number system and place value

Planning Notes:

- Review how programming software handles character representations.
- Have eight pennies for each pair of students and either a handout and/or overhead of bit information
- Review binary and base 10 conversions.
- Prepare coded messages for the students to decipher.
- Have copies of ASCII code available (both standard and extended).

Teaching/Learning Strategies:

- Show segment 3 of The Journey Inside video (8 min 25 sec—Intel Corporation. The Journey Inside. Part of The Journey Inside Education kit), or any other video that shows how computers turn pictures and colors into codes. Students gain an understanding of how information is communicated through the use of codes.
- Hand each pair of students eight pennies and work through the questions on bit information. Ask students what pattern they can see forming in the right column (numbers double).
- Students are challenged to count as high as they can on one hand and told the answer is greater than 10. While students ponder the challenge, teachers demonstrate, with the aid of a simple series circuit, the binary logic states of ONE and ZERO (TRUE and FALSE, HIGH and LOW) by equating them to series circuit lamp ON and OFF condition.
- Binary numbers are introduced by initiating finger counting on one hand—no fingers up is zero, thumb up is a one, index finger up is two, middle finger up is a four, ring finger is eight, and pinkie finger represents sixteen. Students demonstrate counting to 31 on one hand.
- This sets the stage for demonstrating how to convert numbers from Base 10 to Base 2 (binary). Work through several examples with students.
- Give students a quiz on binary conversion to assess their grasp of the concept.
- Handout the ASCII conversion information. Since computers cannot think like we do, they need a code

to translate our language into data that they can process and then convert that data back into recognizable language.

- Students complete conversion exercises.

Assessment/Evaluation Techniques:

- Formative assessment of quiz at the end of the binary conversion exercise to prompt students on progress and show changes required for success of conversion application.
- Summative assessment of conversion exercises.

Accommodations:

- Use extensive visual aids and demonstrations to assist students as needed.
- Provide an enlarged copy of conversion methodology in classroom as well as ASCII character chart.
- Use a variety of teaching styles to accommodate learning styles.
- Provide appropriate adaptive devices or implementation accommodations for identified students.

Resources: . . . . . Adapted from the course profile for Computer Engineering Technology, Grade 10, Unit 2: Integrated Circuits (page 53) Ontario Ministry of Education (*www.acse.net/resources.htm*)

Activity: . . . . . Setting up a Computer

Time: . . . . . 2½ hours

Description: . . . . . Students set up a computer including installing available software and an operating system. Students connect, configure, and test all peripherals. Finally, students troubleshoot any problems that arise. All students set up a PC.

Level: . . . . . II (Grades 9-10)

Topics: . . . . . 1—Students will gain a conceptual understanding of the principles of computer organization and the major components (input, output, memory, storage, processing, software, operating systems, etc.).

Prior Knowledge: . . . . . Components of a computer system, correct terminology

Planning Notes:

- Prepare available samples of micro-controllers and PCs of various types.
- Determine the most effective use of existing hardware within the recommended time allotment (e.g., two to three students per computer).
- Open an older discarded hard drive for demonstration purposes.
- If resources are limited, a single system may be set up several times to accommodate all students.
- This activity is done with stand-alone machines to not interrupt a networked environment.
- The teacher should review the procedures in the attached appendices. This activity assumes that the computer system hard drive has been configured prior to the installation of the operating system.
- The actual system installation can be performed as a class "walk through." The teacher can modify the process to have the individual groups perform the set-up task.
- The teacher should review the disk partitioning, formatting, scandisk operations, and information available in the Help files of the operating system (see Resources).
- Inventory the operating system CD-ROM and software key.
- Ensure all software is available for the full installation including operating systems, device drivers, and application software.

Teaching/Learning Strategies:

- Teachers and students review safety with static electricity and the importance of keeping contacts clean as they apply to components. Review the safety considerations when setting up a desktop computer (grounded plugs, using power bars, dangling cords, eliminating the danger of static electricity, and unplugging power supply before opening a PC, etc.).
- The teacher explains how hard drives work so that students can understand the utility functions they are required to complete by the end of the activity.
- Students use the equipment they require to complete the task, including the monitor, CPU, keyboard, mouse, and a printer, if available. The teacher explains any special considerations they need to know (e.g., positioning of computers for plugs in the room). Students use this information to create a checklist for the activity.
- Depending on the resources available, divide students into the appropriate number of groups. Students connect all the parts of their computer system. Circulate to help with troubleshooting and use questioning techniques to assist with problem solving.
- Once all components are connected, students load the operating system software. Students complete their personal checklist to keep in their portfolios.
- All groups must then test their software to ensure their system is working and that all peripherals connected are functioning properly.

Assessment/Evaluation Techniques:

- A formative assessment through student discussion and observation, encouraging students to assess their thinking for successful completion of task.
- Assess student-created checklists. Provide students with written/oral feedback, to assist their success in upcoming related activities.

Accommodations:

- Provide step-by-step instructions.
- Provide a glossary of terms.
- Provide visuals of different computer types.

Resources: Course Profile: Computer Engineering Technology, Grade 10, Unit 3: Networking (page 72) Ontario Ministry of Education (*www.acse.net/resources.htm*)

Activity: Careers in Computer Engineering

Time: 3¾ hours

Description: A guest speaker is invited to share information about his/her job/career with the students. Students expand on their computer industry knowledge. Students look at degrees and certifications available and opportunities they have at the high school level and beyond to move them toward careers in the computer industry.

Level: II (Grades 9-10)

Topic: 10—students will gain a conceptual understanding of the 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).

Prior Knowledge: . . . . . Word-processing skills

Planning Notes:

- Guest speakers may include the school sysop, board technician, or someone from the local community.
- Collect information from a local university or community college, including school course calendars and college/university catalogues.
- Gather copies of recent computer trade magazines.
- Arrange ahead of time for a student to introduce guest speakers and another student to thank them.
- Collect newspaper advertisements for jobs in the computer industry.
- Distribute a sample certification worksheet

Teaching/Learning Strategies:

- Teachers introduce the expectations of the activity.
- Teachers review with students (ahead of time) questioning techniques for the guest speaker.
- One student may introduce the guest speaker. Students take brief notes in order to ask relevant and interesting questions. One student may thank the guest speaker.
- Discuss the speaker information with the students, after which they write their personal views on the information.
- Students look through trade magazines to see advances in the computer industry. Each student picks one article from a magazine to summarize or review using a word processor.
- Finally, students look at opportunities for different computer designations ranging from MCSE (Microsoft Certified Engineer) to computer engineering at the university level. Students use newspaper advertisements to explore what skills and designations are requested by potential employers.
- Students retrieve the certification chart file (either electronically or via handout) and, using designations discovered in the advertisement exercise above, they complete the chart and add it to their portfolio.
- Students create a plan on how to pursue a computer career, beginning with the completion of this course, and save the information in their portfolio (long-term goal).

Assessment and Evaluation:

- Review of student portfolio to provide written/oral feedback on completion and comprehension of tasks given.
- Evaluate the article review using the rubric provided.

Accommodations:

- Allow flexible timelines for due date of report.
- Use career center videos if available.
- Invite the Student Services resource personnel into the classroom.
- Videotape the guest speaker(s) presentation to allow students an opportunity to watch it again.

Resources:. . . . . Course Profile: Computer Engineering Technology, Grade 10, Unit 3: Networking (page 93), Ontario Ministry of Education (*www.acse.net/resources.htm*)

Activity: . . . . . Connections Inside and Out

Time: . . . . . 3⅓ hours

Description:. . . . . Students view the video The Journey Inside The Computer (from Intel Corporation (*http://secure.wesweb.com/intel/form.htm*) and examine the

individual internal components of the computer. Using resources available to them, students discover the importance of each component and its impact on the computer's operations. The activity culminates with a series of problems that students must solve using the new knowledge. Finally, students use this information to suggest an alternative placement of computers within the school environment that makes a positive impact on the school community and demonstrates wise use of resources.

Level: . . . . . II (Grades 9-10)

Topics: . . . . . 1—students will gain a conceptual understanding of the principles of computer organization and the major components (input, output, memory, storage, processing, software, operating systems, etc.).

3—students will gain a conceptual understanding of the basic components of computer networks (servers, file protection, queues, routing protocols for connection, communication, spoolers and queues, shared resources, and fault-tolerance).

Prior Knowledge:

- the differences between hardware and software;
- ability to record findings from observation;
- familiarity with the operating system they are using and the term *network;*
- familiarity with internal components and their uses.

Planning Notes:

- Request permission for students to visit certain areas of the school during class time—plan this as an in-school field trip.
- Think of visiting a music midi lab, communication lab, front office, and any specialized resources specific to your local environment.
- Check with the site administrator if you are not sure of network type(s) available in the school.
- Prepare checklist of terms for student use during video.
- Arrange to have a computer site administrator from the school or board office or a computer technician speak to the class about networks and operating systems
- Have a school map available for students to take on tour and an overhead of the map for review.
- Check for materials from The Journey Inside The Computer kit available from Intel (Intel Corporation. The Journey Inside. Part of The Journey Inside Education kit.)

Teaching/Learning Strategies:

- show The Journey Inside The Computer video, Unit 4 on Microprocessors, then Unit 6 on Networking, with the purpose of reviewing computer components and extending student knowledge of networks and operating systems;
- take up terms sheet and have students complete definitions for words they are unfamiliar with (teachers may introduce students to the online dictionary at www.dictionary.com);
- share information on networks with students;
- indicate type(s) of networks currently used in the school environment;
- share information on operating systems with students;
- deliver short test on networks and operating systems;
- provide each student with a map of the school and explain tour route and any special routines required for secure areas;

- give students a simple key for marking on map (e.g., C = stand alone computer, L = lab, SL = specialized lab, S = server room, P = printer resource);
- return to the classroom and review the map on an overhead with input from students;
- encourage a discussion of how improvements that have been made in network and operating systems make a difference in a computer community such as a school;
- ask them to reflect on why they think the computer resources have been placed in the school the way they are;
- ask students to prepare a written brief of changes they would like to see in the school computer environment;
- direct students to include positive impact(s) their suggestions have on the school environment and incorporate their knowledge of networks;
- facilitate student pair/square and share of suggestions.

Assessment/Evaluation Techniques:

- a formative assessment in use of the review terms sheet, and
- an evaluation of test on networks and operating systems.

Accommodations:

- Give an oral test if appropriate;
- provide students with physical disabilities assistance if required;
- assist students with special needs with terms sheet.

Resources: . . . . . Course Profile: Computer and Information Science, Grade 10 Unit 4: The Computer and Society (page 116), Ontario Ministry of Education (*www.acse.net/resources.htm*)

A.3 Sample Activities for Level III: Computer Science as Analysis and Design

Activity: . . . . . New Solutions for Old Problems

Time: . . . . . 5 hours

Description: . . . . . Students examine problems that can be solved using more than one algorithm (e.g., determining the factorial value of a number). Using brainstorming or other group problem-solving techniques, students develop alternative algorithms using recursive and non-recursive techniques. Students identify the components of a recursive algorithm and develop criteria for recognizing when a recursive algorithm may be applied.

Level: . . . . . III (Grades 10-11)

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.

Prior Knowledge and Skills: . . . . . use of problem-solving models, the ability to develop appropriate algorithms to solve problems, and the ability to write pseudocode.

Planning Notes:

- Review the nature of recursion.
- Gather examples of problems that can be solved using more than one method, including recursion, and determine which problems may be solved using a recursive algorithm.

Teaching/Learning Strategies:

- divide the class into groups of two or three students.
- review the brainstorming problem-solving technique.
- present a problem that can be solved using a familiar but complex algorithm and may also be solved using a less familiar but simpler algorithm (e.g., determining the quotient and remainder of the division of two integers).
- Students, in their groups, develop more than one algorithm for the solution.
- The teacher facilitates a class discussion to develop criteria for the evaluation of algorithms, including the efficiency of the solution and the complexity of the required coding. Both processing and user interface efficiencies are considered.
- Groups evaluate the algorithms using the developed criteria and share their algorithms and evaluations with the class.
- The teacher introduces the recursive method of problem solving and illustrates a recursive algorithm for the solution to a different problem (e.g., calculating the factorial value of a number).
- Groups develop a recursive algorithm to the initial problem and evaluate its efficiency.
- The teacher facilitates a class discussion to establish criteria for determining if a recursive algorithm is an appropriate solution and identifies additional problems that may be solved using recursion.
- Working in groups, students develop recursive and non-recursive algorithms for additional, assigned problems.

Assessment and Evaluation:

- A formative assessment of the assigned in-class work in the form of roving conferences, and
- a summative assessment in which students complete an assignment requiring the development of both a recursive and a non-recursive algorithm.

Accommodations: . . . . . Provide print copies of examples of algorithms using recursive and non-recursive methods, including graphic illustrations, and use models to illustrate the algorithms.

Activity: . . . . . Planning a Solution

Time: . . . . . 6 hours

Description: . . . . . Students work in groups to analyze complex problems (e.g., Towers of Hanoi) and to develop appropriate algorithms using recursive and non-recursive techniques. Students create pseudocode and design charts to assist them in planning a solution and assess these representations of code as problem-solving tools.

Level: . . . . . III (Grades 10-11)

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 and 7—principles of software engineering: software projects, teams, the software life cycle.

Prior Knowledge and Skills: . . . . . students can apply the steps in the software design life cycle; use pseudocode, diagrams, and charts to summarize program design; and develop appropriate algorithms to solve problems.

Planning Notes:

- Review top-down problem solving.
- Select a problem to use in developing a model solution and prepare the appropriate models.

Teaching and Learning Strategies:

- The class is divided into groups of two or three students and each group is assigned a problem.
- Groups investigate the problem, using a variety of problem-solving techniques to analyze it.
- Each group uses brainstorming or other group problem-solving techniques to develop an algorithm for the solution to the problem. In a class discussion, groups present and share their algorithms.
- Students compare the effectiveness and efficiency of the algorithms presented and then the groups refine their algorithms.
- Each group develops a flow chart, structure chart, and/or pseudocode to represent the application of the algorithm. The teacher conferences with each group to discuss and assess the solution design.

Assessment and Evaluation:

- a formative peer assessment of the presented algorithms;
- a formative assessment of the design for the solution to the problem.

Accommodations:

- Provide print sample algorithms similar to the one studied.
- Use graphical models to illustrate the problem.
- Selectively pair/group students to assist problem solving.
- Provide problems of varying complexity to provide an appropriate challenge.

Activity: . . . . . Role Playing Helper Functions/Recursion
Time: . . . . . 1 hour
Description: . . . . . Students role play various objects of simple programs to understand parameter passing and recursive calls
Level: . . . . . III, IV (Grades 10-12)
Topics: . . . . . 1—methods (functions) and parameters, recursion
Prior Knowledge and Skills: . . . . . compile and run simple programs; write code using parameters.

Planning Notes:

- Prepare or obtain from resources scripts for role playing objects in a small program with several nested (and usually also recursive) calls.
- Gather colored markers and poster board as needed.

Teaching/Learning Strategies:

- Review the concepts of constructors, parameters, and calling helper methods.
- Students read code for the program to be used for the role play.
- Select a student to role play the main function. If desired, give student a large name tag to wear. Select another student to be the code monitor whose job is to keep a record of the current line of code being executed.
- Assist class as they act out the script, each time an object is constructed the student calling the constructor function picks a classmate to play the role of the object; if using name tags, be sure to give each object-player a name tag. Using different sized, shaped, or colored tags for different classes is helpful.
- Frequently pause the play and ask audience members to identify who the next actor will be.

Assessment and Evaluation:

- a formative assessment of the role play in the form of roving interviews;
- a summative assessment can be administered asking students to indicate the number of objects in existence as the play progressed and similar questions.

Accommodations:

- Let pairs of students play a role.
- Assign roles yourself giving simpler roles to students who are struggling.

Resources: . . . . . Several role playing exercises are available at: *http://cs.colgate.edu/APCS/Java/RolePlays/JavaRolePlays.htm*

A.4 Sample Activities for Level IV: Topics in Computer Science

Activity: . . . . . Introduction to Object Oriented Design

Time: . . . . . 3 hours

Description: . . . . . Students are introduced to the initial steps of applying Object Oriented Design to a programming problem and practice applying those steps to a problem that may later be used as a significant programming project

Level: . . . . . III, IV (Grades 10-12)

Topics: . . . . . 1—fundamental ideas about the process of object oriented program design

Prior Knowledge and Skills: . . . . . none

Planning Notes:

- Prepare blank diagrams for use as CRC cards and/or object diagrams
- Select examples of problems that can be solved by using object oriented techniques.

Teaching/Learning Strategies:

- Explain the differences between an object oriented and a functional approach to the design of a computer program.
- Have a student explain how a card game (like blackjack) is played.
- Working with the class, identify potential objects involved in the game.
- Working with the class, identify possible operations that might be done by or to a card or one possible object. Students in small groups brainstorm operations for other objects identified as part of the problem.
- Illustrate how the two notations can be used to summarize the analysis so far.
- Discuss possible relationships between objects.
- Show notations for relationships.
- Describe a possible scenario in the game and use developed notations to represent that scenario.
- Have students in pairs describe a second scenario and represent it.
- Share class scenarios and consider whether they collectively represent the range of possibilities especially extreme scenarios.
- Students read the possible problems and select one on which to do an OOD. Students work individually on the first two phases (identify objects and operations).
- Students working on the same problem share results and agree on a "best" set of objects and operations.
- Students individually complete the last two steps.

Assessment and Evaluation:

- a formative assessment of the assigned in-class work in the form of roving conferences;
- a summative assessment applying OOD to a new problem

Accommodations: . . . . . Provide copies of problem descriptions, with important nouns and verbs indicated using a different font or type size. Provide written scenarios for each problem.

Resources:

- Wirfs-Brock, Wilkerson, and Wiener, *Designing Object-Oriented Software*, Prentice Hall, 1990
- Fowler, *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley, 1997.
- Overview of lesson and sample exercises are available at:
- *http://max.cs.kzoo.edu/AP/OOD/OODPresentation/*
- *http://max.cs.kzoo.edu/AP/OOD/OODSpecifications/*

A.5 Additional Resources for Level IV: Topics in Computer Science

A wide range of resources is available for supporting a Level IV computer science curriculum.

AP Computer Science course

The curriculum for this course is governed by the topic outline available from The College Board at *http://www.apcentral.collegeboard.com/*. Two example activities to enhance learning for the AP curriculum are shown earlier in this Appendix.

Courses Leading to Industry Certification

Many of the certification courses provide a prepared curriculum that details the content and order of topics. While implementation of this type of course may be simplified by the information provided, careful evaluation is needed with regard to proprietary content versus general concepts. Information about the content of some certification courses is available at *http://www.computer-certification-training.com*.

Project-Based Courses

Project-Based courses can provide targeted education geared toward specific student interests in Computer Science. While these courses can be offered by the local school district, enrolling students in a college-based course should be considered. Computer project-based courses are often offered in a college or university department of computer science, information technology, or information systems. Computer-based programs of study at the college level are typically well established and provide a variety of current topics. Additionally, a student participating in a college computer course may be eligible for college credit. The secondary school will need to investigate the content of each course to determine if it is appropriate for Level IV study.

Courses for college credit can be offered on the college campus, at the high school, or via distance education. A course offered on the college campus will place less burden on the high school for supporting special hardware, software, and faculty resources.

Online courses can be taken off-hours or during school time. A student who chooses to take an online course outside of school hours will need access to specified hardware, software, and the Internet. When a college-delivered online course is offered during the school day, it may be possible to schedule several online courses during the same class session, allowing better utilization of faculty time. To support this, the school must also provide access to the required hardware, software, and the Internet.

A course for college credit offered at the secondary school permits the student and high school faculty member to interact in a traditional manner. The secondary school will need to ensure that the curriculum is sanctioned by the college and students are officially enrolled in the college. The college may wish to participate in the selection of the faculty member delivering the course.

Some typical methods by which high school students can achieve college credit are described below.

Tech Prep and Articulation Credit—The student takes a course at the high school and receives high school credit. The course is also pre-approved for college credit at a particular college, and is typically taught by a high school faculty member. Credit is awarded for the course upon matriculation at that college. The college may require that the student pass a competency exam before applying the credit to the student's transcript.

Dual Credit / Concurrent Enrollment—The student is enrolled in a course for which s/he simultaneously receives high school and college credit. The student must meet all college requirements for entrance into the course.

Challenge Exams—The student may be able to prove proficiency and receive credit by exam. This method is useful for a student who completes a high school course that is not articulated, or who believes s/he has independently gained knowledge of all topics covered by a specific course. The student may be charged a fee to sit for the exam.

Advanced Placement / CLEP Test—A student planning college study toward a career in Computer Science, Information Technology, or Engineering may wish to take the AP Exam in Computer Science. A student planning college study toward a career in Business may wish to take the CLEP Exam in Information Systems and Computer Applications. It is recommended that the student determine how credit will be granted for success on the exam from his/her targeted institution.

Programs That Provide College Credit to High School Students:

A number of programs provide college credit to high school students. Here are three examples:

The University of Pittsburgh's College in High School is one example of high schools interacting with a college. The "College in High School (CHS) program has offered qualified high school students the opportunity to earn University of Pittsburgh college credits during their regular school day." Students are required to pay a reduced tuition to participate in the program. Financial Aid may be available. The program offers "12 courses to over 2,600 students in 100 high schools with 224 faculty" (Pittsburgh). There is an extensive list of colleges that will accept CHS credits for transfer at *http://www.pitt.edu/~chsp/transfer.htm*.

The University of Cincinnati also offers high school students the opportunity to earn college credit by providing a Post-Secondary Enrollment Options Program (PSEOP). Currently there are "39 high school students enrolled in the program, including one freshman, three sophomores, 13 juniors and 22 seniors". This program permits the student to enroll in the college for college credit only, or to enroll for both high school and college credit.

The University of Northern Colorado provides "High School Concurrent Credit, a program for Colorado Residents enabling high school Juniors and Seniors to earn college credit while in high school". Three options are provided under the program:

> "**Option I**—Fast Track Program: For the student who is a high school senior and has met high school graduation requirements.
> **Option II**—Post-Secondary Enrollment Options Program: For the student who is a high school junior or senior and has not met high school requirements.
> **Option III**—College Acceleration Program: For the student who is a high school junior or senior and wants to accelerate his or her college program whether or not the graduation requirements have been met."

Resources:

Seven Ways to Earn College Credit in High School, *http://www.careersprep.com/html/sevenwys.html*

Arthur R. Greenberg, ERIC Clearinghouse on Higher Education Washington DC, BBB27915, George Washington Univ. Washington DC. School of Education and Human Development., ERIC Identifier: ED347956, Publication Date: 1992-03-00, *http://www.ericfacility.net/ericdigests/ed347956.html*

Comparison of Methods to Receive College Credit for Courses Taken in High School, *http://www.tea.state.tx.us/Cate/teched/collegecreditinhs.pdf*

Dawn Fuller, University of Cincinnati, High School Students: Do You Qualify for UC College Credit?, March 7, 2003, *http://www.uc.edu/news/NR.asp?id=300*

The College Board, AP Central, *http://apcentral.collegeboard.com/*

University of Cincinnati, Post Secondary Enrollment Option Program, *http://www.esit.uc.edu/pseop/psoDefault.aspx*

University of Northern Colorado, High School Concurrent Credit, Last Update: August 5, 2003, *http://www.unco.edu/admissions/collegecredit.html*

University of Pittsburgh, College in High School, Last Update: June 25, 2003, *http://www.pitt.edu/~chsp/index.htm*

U.S. Department of Education, Dual Enrollment, Last Update: January 27, 2003, *http://www.ed.gov/offices/OVAE/CCLO/dualenroll.html*

U.S. Department of Education, Program Title: Tech-Prep Demonstration Program, CFDA # 84.353, *http://web99.ed.gov/GTEP/Program2.nsf/02cbabc638062ed2852563b6006ffeae/0c81ea75d203e37d85256a01006527b1?OpenDocument*

U.S. Department of Education, Tech Prep Education, Last Update: January 28, 2002, *http://www.ed.gov/offices/OVAE/CTE/techprep.html*

The first year of your CSTA membership is *FREE*!

WHAT IS THE COMPUTER SCIENCE TEACHERS ASSOCIATION (CSTA)?

The Computer Science Teachers Association, a limited liability company under the auspices of ACM, has been organized to serve as a focal point for addressing several serious (crisis level) issues in K-12 computer science education, including:

- Lack of administrative, curricular, funding, professional development and leadership support for teachers
- Lack of standardized curriculum
- Lack of understanding of the discipline and its place in the curriculum
- Lack of opportunities for teachers to develop their skills and interests

The above issues result in:

- A profound sense of isolation, and
- Dropping enrollment in college level computer science programs

There are other organizations that address use of technology across the curriculum, but only CSTA speaks directly and passionately for high school computer science.

OUR MISSION

CSTA is a membership organization that supports and promotes the teaching of computer science and other computing disciplines at the K-12 level by providing opportunities for teachers and students to better understand the computing disciplines and to more successfully prepare themselves to teach and to learn.

OUR GOALS

CSTA's organizational and educational goals include:

- Helping to build a strong community of CS educators who share their knowledge
- Providing teachers with opportunities for high quality professional development
- Advocating at all levels for a comprehensive computer science curricula
- Supporting projects that communicate the excitement of CS to students and improve their understanding of the opportunities it provides
- Collecting and disseminating research about computer science education
- Providing policy recommendations to support CS in the high school curriculum, and
- Raising awareness that computer science educators are highly qualified professionals with skills that enrich the educational experience of their students

OUR SCOPE

The scope of the organization includes:

- High school (all aspects of computer science education)
- Elementary and middle school (introducing problem solving and algorithmic thinking)
- College/university (to establish better transition for high school programs and provide a greater level of support to high school teachers)
- Business and industry (supporting computer science education and teachers)

WHO SHOULD JOIN?

- All High School & Middle School Computer Science Teachers
- All K-12 Computer Applications Teachers
- All individuals interested (and passionate) about K-12 CS Education

The first year of your CSTA membership is *FREE*!

JOIN US VIA...

web: *http://csta.acm.org/*
phone: 1.800.342.6626
e-mail: *cstahelp@acm.org*

2 Penn Plaza, Suite 701, New York, NY 10121-0701

Realizing its commitment to K-12 education