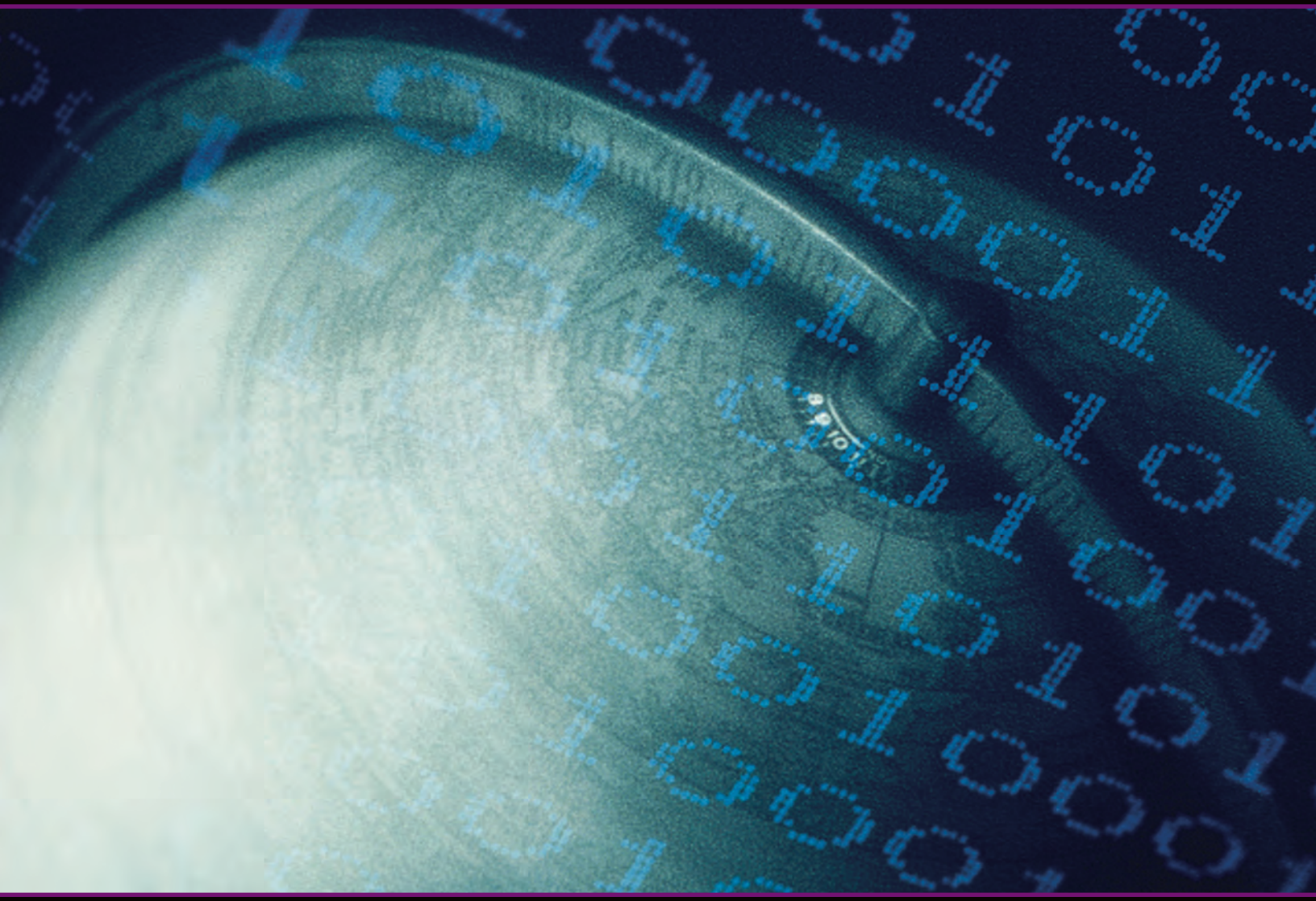

The New Educational Imperative:
Improving High School Computer Science Education

Using worldwide research and professional experience to improve U. S. Schools

By CSTA Curriculum Improvement Task Force

csta Computer Science Teachers Association

The New Educational Imperative:

Improving High School Computer Science Education

Using worldwide research and professional experience to improve U. S. Schools

By The CSTA Curriculum Improvement Task Force

csta Computer Science Teachers Association

The New Educational Imperative: Improving High School Computer Science Education

Final Report of the CSTA Curriculum Improvement Task Force

February 2005

Task Force Chair

Chris Stephenson
CSTA Executive Director

Committee Members

Judith Gal-Ezer
Open University of Israel

Bruria Haberman
Holon Academic Institute of Technology
and The Weizmann Institute of Science

Anita Verno
Bergen Community College

Computer Science Teachers Association
Association for Computing Machinery
1515 Broadway, 17th Floor
New York, New York 10036

ACM ISBN: # 1-59593-335-2
ACM Order Number: # 999064

Additional copies may be ordered prepaid from:

ACM Order Department
P.O. Box 11405
Church Street Station
New York, NY 10286-1405

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

This material is based upon work supported by the National Science Foundation under Grant No. 0455403. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Letter from the CSTA President

In the face of a global economy that rewards innovation and technological supremacy, many countries, including the United States, are looking to education to ensure their continuing competitiveness.

With the specific goal of keeping the United States technologically competitive in fields such as nanotechnology, supercomputing, and bioinformatics, the United States has committed to training thousands of high school teachers to lead advanced placement math and science courses.

Computer science as an academic discipline provides the knowledge and skill foundation for all of these technological advances.

Computer science education in the United States is at a critical juncture. The number of students pursuing computer science as a career choice or course of study has dropped precipitously at all levels. The number of high school students taking the Advanced Placement computer science exam, for example, has declined almost 20% over the past three years. In addition, the lack of national curriculum standards and consistent and rational teacher certification requirements continue to hamper our ability to ensure that our students are adequately prepared to compete in this increasingly technological world.

In short, policy makers at the local, state, and national levels must prepare for a 21st century technological workforce by giving those workers the computer science knowledge base necessary for success in these computing-intensive fields.

The warning signs are unmistakable. The solution is clear. Unless we act quickly and decisively to remedy the disconnect between our national technological goals and computer science education at the high school level, the United States will soon face an educational, competitive, and economic crisis.

I urge you to read closely and carefully this final report of the CSTA Curriculum Improvement Task Force and heed its sobering message. Improving high school computer science education is indeed a national educational imperative.

Regards,

Robb Cutler
Chairperson
Computer Science Teachers Association

Acknowledgments

The CSTA Curriculum and Improvement Task Force would like to thank the following organizations and individuals. First, we would like to thank the National Science Foundation for their generous support of this project and for their understanding of the importance of improving computer science education at the high school level.

We would also like to thank Allen Tucker and the ACM K–12 Curriculum Committee for their ground-breaking work on the *ACM Model Curriculum for K–12 Computer Science* and Jane Margolis and Joanna Goode for sharing the important work that they are doing as part of the Computer Science Equity Collaborative.

We are especially grateful to the wonderful international educators who took part in our panel on creating and implementing an effective curriculum, specifically Michael Chiles, Judith Gal-Ezer, and Jackie Martin.

Thanks are also due to our wonderful designer Bob Vizzini, and to our exacting copy editor, Kate Conley.

Finally, to all of the staff and volunteer leadership of ACM, who brought CSTA into being and continue to support us every day.

contents

The New Educational Imperative:
Improving High School Computer Science Education

Final Report of the CSTA Curriculum Improvement Task Force

February 2005

Executive Summary

THE ECONOMIC AND EDUCATION IMPERATIVE

The United States, a nation once proud of its leadership in education, is sitting quietly on the sidelines while other countries make improvements to ensure their high school graduates will be ready to meet the demands of tomorrow's high-tech society. Countries around the world that once looked to the United States for guidance when planning new and innovative curricula are now taking the lead in computer science education. While other countries are requiring a computer science course for high school students just as they require math or biology, high school computer science education in the United States is disappearing. In light of the anticipated shortage of qualified candidates for the 1.5 million computer and information technology jobs by 2012, this lack of engagement with issues relating to computer science education is shortsighted and potentially disastrous.

DEFINING COMPUTER SCIENCE

Unlike other more static disciplines, computer science is constantly being reshaped. New thinking and new technologies continue to expand our understanding of what computer scientists can and need to know. This has resulted in considerable debate about a single definition of computer science. The ACM *Model Curriculum for K–12 Computer Science,* however, provides the most appropriate definition of computer science for high school educators. Computer science, it argues, is neither programming nor computer literacy. Rather, it is "the study of computers and algorithmic processes including their principles, their hardware and software design, their applications, and their impact on society" (Tucker, 2003). Computer science therefore includes:

- 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 computations (what computers can't do),
- applications in information technology and information systems, and
- social issues (Internet security, privacy, intellectual property, etc.).

HIGH SCHOOL COMPUTER SCIENCE EDUCATION IN THE UNITED STATES

Computers have infiltrated all areas of society, and there is now a clear link between technology, innovation, and economic survival. In light of this, one would expect a move within our society to support and standardize computer science education.

Yet, no national K–12 computer science curriculum exists. Lack of leadership on high school computer science education at the highest legislative and policy levels has resulted in insufficient funding for classroom instruction, resources, and professional development for computer science teachers. In addition, complex and contradictory teacher certification requirements as well as salaries that cannot possibly compete with industry make it exceedingly difficult to ensure the availability of exemplary computer science teachers.

In the face of confusing definitions of computer literacy, information fluency, and the various sub-branches of computer science itself, many schools have lost sight of the fact that computer science is a scientific discipline and not a "technology" that simply supports learning in other curriculum areas. Computer science is not about point and click skills. It is a discipline with a core set of scientific principles that can be applied to solve complex, real-world problems and promote higher-order thinking. In short, knowledge of computer science is now as essential to today's educated student as any of the traditional sciences.

WHAT THE RESEARCH REVEALS

The body of research from around the world relating to high school computer science education indicates that learning computer science provides direct benefits to students. While there are distinct differences between how various countries implement their high school computer science programs, a growing number of countries already require computer science education of all high school students.

The primary limitation of the current body of research is that it is too narrowly focused on programming and therefore does not give an appropriately broad view of the discipline. Despite this limitation, however, the research provides vital information regarding students' misconceptions about computer science (for example, that it is just about programming or that it is a "male" field). It also provides critically important insights about the underlying principles of computer science education.

- Students should acquire a broad overview of the field to construct a comprehensive picture of computer science as a discipline.
- Students must understand not only the theoretical underpinnings of the discipline but also how that theory influences practice.
- Computer science instruction should focus on problem solving and algorithmic thinking.
- Concepts should be taught independent of specific applications and programming languages.
- Students should be taught what will be expected from them in the "real world," specifically, what is actually required to write and maintain computer programs and large software systems. Computer Science should be taught using real-world applications rather than specialized educational tools.
- Computer science instruction should include integrative and interdisciplinary knowledge.
- High school students should be exposed to advanced topics of computer science (e.g. computational models, modeling, and parallel computing) to enable them to become familiar with some of the theoretical aspects of computer science.
- Students must recognize recurring concepts and principles such as abstraction, complexity, modularity, and reusability.
- Programming should be taught in a broad sense, covering not only the coding act itself, but also the design of the algorithms underlying the programs and, to some extent, considerations of correctness and efficiency.
- The curriculum should be designed to address the under-representation of women and minority students in computer science.
- Teachers should motivate students to endure the rigors of traditional computer science programs by engaging students with exciting, accessible, and leading-edge courses.
- Teaching and learning activities should be

designed to treat common misconceptions of the essence of computer science.

The research also indicates that student success in computer science is predicated on the teacher's knowledge of the discipline and ability to actively engage students in their own learning, and identifies several key factors that help to ensure exemplary teaching.

- High school computer science teachers must have a thorough formal background in computer science.
- Pre-service teacher education must prepare teachers to better employ general pedagogical principles as well as teaching methods in the context of computer science education.
- To be best qualified, computer science high school teachers should be certified and offered courses in computer science education in addition to regular computer science courses.
- Classroom teachers require on-going access to appropriate and relevant professional development opportunities that allow them to master new technologies, implement new curricula, and constantly improve their teaching.
- Teachers should become part of a collaborative computer science teachers' community of practice by joining local and national associations that provide useful resources and support their ongoing learning and leadership development.

WHAT WE CAN LEARN FROM OTHER COUNTRIES

Looking at the experiences of other countries that have designed and implemented a high school computer science curriculum can provide valuable lessons as to the factors that must be in place to ensure that our efforts to improve computer science education are successful. An examination of the experiences of an international panel of computer educators from Canada, Israel, Scotland, South Africa, and the United States support the argument that such curriculum initiatives will be successful only to the extent to which they meet the following criteria:

- There is a link between the outcome required and the strategies used.
- Change is driven by real learning needs and not politically manufactured needs.
- Educational change must be seen in the context of larger social and economic forces.
- All of the stakeholders must agree to the need for change and on the strategies put in place to achieve it.
- Change requires the commitment of adequate resources through all phases of the design, implementation, and testing of the new curriculum.
- Change is a long-term process, not a short-term intervention.

ACTIONS TO IMPROVE HIGH SCHOOL COMPUTER SCIENCE EDUCATION

Education in the United States is at a crossroads. We can either commit to ensuring that our students have the skills to be participants and innovators in this technological world or resign ourselves to a decreasing international presence in the global economy. We know that the world continues to change and that many of the changes we face are related to the burgeoning possibilities and consequences of our growing dependence upon and interrelation with computing technology. Maintaining our ability to meet present and future challenges requires us to acknowledge computer science as a core element of all STEM (science, technology, engineering, and mathematics) initiatives. Sustaining our technological and innovative edge in this increasingly global economy requires a multi-level, nationwide commitment to improving computer science education at the high school level. We must make this commitment if our schools are to continue to provide relevant education and our society is to continue to solve problems on the cutting edge of innovation. Here is what we need to do:

- We must begin with a definition of computer science education that recognizes computing as a scientific discipline and encompasses a breadth of knowledge and skills.
- We must implement a national computer science curriculum for high schools. This curriculum must be principle-based, must address core content and key skills, and must incorporate appropriate strategies to reach and teach our students.
- We must support the implementation of this new curriculum with a plan that includes a realistic timeframe and the provision of the resources required to achieve it.
- We must ensure that computer science is taught by exemplary teachers who have the requisite knowledge to teach the curriculum and who continue to upgrade their technical and teaching skills throughout their careers.

Schools alone cannot achieve these outcomes. This issue requires a national vision, supportive action, and commitment at all levels of the political and educational systems. The final chapter of this document therefore provides practical, achievable suggestions for ways in which

- federal government policy makers,
- state government policy makers,
- school district policy makers,
- school principals,
- teachers,
- university and college faculty, and
- business and industry leaders

can work together to achieve long-term, systemic improvements to high school computer science education.

ORGANIZATION OF THIS DOCUMENT

This document has been designed to address a number of different aspects of high school computer science education. Chapter One (*High School Computer Science in the United States*) provides an overview of the current state of high school computer science education. This chapter discusses the ways in which the United States has failed to support and standardize computer science education despite the infiltration of computing technology into all areas of society.

Chapter Two (*Challenges and Perspectives in High School Computer Science Education: A Retrospective Literature Review*) provides an extensive review of international research literature in the area of high school computer science education, particularly as it relates to course content and pedagogy. This chapter highlights the continuing focus on programming despite more recent educational models that suggest that students need a broader theoretical and skills foundation to ensure that they are prepared for college and the workplace.

Chapter Three (*Using Frank's Framework to Explore the Development and Implementation of High School Computer Science Curricula in Five Countries*) looks at the development and implementation of high school computer science curricula in five countries. Using a specific framework for critiquing educational policy reform and implementation, this chapter examines the experiences of curriculum development specialists from Canada, Israel, Scotland, South Africa, and the United States as presented at a panel session at the 2005 National Educational Computing Conference. This chapter provides valuable insight on key implementation issues such as funding, professional development, and time lines.

Finally, Chapter Four (*Strategies for the Successful Design and Implementation of a National High School Computer Science Curriculum*) proposes a set of practical, achievable strategies and recommendations that could profoundly improve high school computer science education in the United States. These changes, requiring the support and commitment of federal and state policy-makers, school district leaders, classroom teachers, university and college faculty, and business and industry, would help provide students with the scientific knowledge base required for success in the 21st century and would ensure that the United States remains competitive in the global economy.

CHAPTER ONE

HIGH SCHOOL COMPUTER SCIENCE EDUCATION IN THE UNITED STATES

1.0 INTRODUCTION

Although an increasing number of labor and economic specialists are beginning to express concern about the direct and pressing link between technology and innovation and national economic survival, very few school administrators and educational policy leaders understand the profound need for computer science education at the high school level. Despite the need for students to incorporate the foundational computer science skills that foster an understanding of the essential technologies found in almost every industry today, in most high schools in the United States, there is little recognition of computer science as a scientific discipline distinct from mathematics or from technology training (National Research Council, 1999; Tucker, 2003)

In today's scientific and economic global communities, computer science is helping to push the boundaries of what we know and what we can do. In areas such as nanotechnology and bioinformatics, computer scientists are addressing key questions such as:

- Why do people become ill?
- How can we do better at feeding the people of the world?
- How can we ensure our own national security and the safety of our people?

As a result of these kinds of scientific breakthroughs, the U.S. economy is expected to add 1.5 million computer- and information-related jobs by 2012.

The problem, however, is that our education system is not producing enough highly skilled people to fill these critical positions. Current projections indicate that this country will have only half that many qualified graduates. As a result, if we do not address these issues immediately and vigorously at the national level, we face long-term skills shortages that will cripple both academic computing and industry (International Technology Association of America, 2002; Sargent, 2004). As a result, the United States will be seriously compromised in its ability to uphold its leadership position in computing, communications, information science, and engineering. In addition, we will be unable to capitalize on current and future innovations that are already providing untapped economic and social opportunities.

1.1 WHAT IS COMPUTER SCIENCE ANYWAY?

One of the challenges we face when discussing computer science education is that the field of computer science seems to evolve so quickly that it is difficult even for computer scientists to clearly define its contents and delimit its boundaries. As Shackelford (2005) noted in his presentation *Why can't smart people figure out what to do about computing education?*, the landscape of computing continues to evolve and trying to figure out what students need to learn is like trying to hit a moving target. While we do know that computing now provides the infrastructure for how we work and communicate and that it has redefined science, engineering, and business, it is still poorly understood.

Even at the university level, where computer science has long been recognized as a key educational and career field, it has resisted a single definition or

application. As Shackelford indicated, prior to 1990 the discipline was defined as *Computer Science* if taught as part of the traditional Arts and Science curriculum and as *Information Systems* if taught as part of the Business curriculum. Since that time, however, it has burgeoned into a number of distinct areas, each of which involves its own approach to both theory and application. These areas include *Computer Engineering*, *Software Engineering*, and *Information Technology*. In terms of university degree programs, the post-1990s are typified by intersecting sub-disciplines including:

- Electrical Engineering,
- Computer Engineering,
- Software Engineering,
- Computer Science,
- Information Technology, and
- Information Systems

that focus to varying degrees on hardware, software, and organizational needs.

For high school educators, though, the most profound confusion arises when trying to distinguish between the three most common kinds of computing education typical of the 9–12 grade levels. While each of these areas has been known by various names, for the purposes of this discussion we define them as:

- Educational Technology
- Information Technology, and
- Computer Science.

In general, *Educational Technology* can be defined as using computers across the curriculum, or more specifically, using computer technology (hardware and software) to learn about other disciplines. For example, the science teacher may use a computer-based simulation program to provide students with a better understanding of specific physics principles, or an English teacher may use word-processing software to help students improve their editing and revision skills.

Tucker, Deek, Jones, McCowan, Stephenson, and Verno (2003) defined *Information Technology* as "the proper use of technologies by which people manipulate and share information in its various forms" (p. 6). While Information Technology involves learning about computers, it emphasizes the technology itself. As Shackleford (2005) noted, Information Technology specialists "assume responsibility for selecting appropriate hardware and software products, integrating those products with organizational needs and infrastructure, and installing, customizing and maintaining those resources" (p. 22). Information Technology courses, therefore, focus on:

- installing and administering computer networks,
- installing, maintaining, and customizing software,
- managing e-mail systems,
- designing web pages, and
- developing multimedia resources and other digital media.

Computer Science, on the other hand, spans a wide range of computing endeavors, from theoretical foundations to robotics, computer vision, intelligent systems, and bioinformatics. According to Shackelford, for example, the work of computer scientists is concentrated in three areas:

- designing and implementing software,
- developing effective ways to solve computing problems, and
- devising new ways to use computers.

Tucker et al. (2003) has also offered a definition of computer science that has direct relevance to high school computer science education. They defined the discipline as follows: "*Computer science (CS)* is the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society" (p. 6). They argue that in order to fulfill this definition, K–12 computer science curricula must include the following 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.).

Citing the conclusion of the National Research Council (1999) that a basic understanding of all these topics is now an essential ingredient to preparing high school graduates for life in the 21st century, Tucker et al. (2003) further argued that the goals of a K–12 computer science curriculum are to:

- introduce the fundamental concepts of computer science to all students, beginning at the elementary school level,
- 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),
- 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, and
- increase the knowledge of computer science for all students, especially those who are members of underrepresented groups.

Two other terms that often appear in discussions of computing education are *Information Technology Literacy* and *Information Technology Fluency* (National Research Council, 1999). As Tucker et al. (2003), indicate:

> 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 … 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 (p. 6).

1.2 WHAT IS HAPPENING IN COMPUTER SCIENCE EDUCATION?

At present, all evidence points to a crisis in computer science education at the high school level. This crisis is most clearly manifested in the decreasing number of computer science courses being offered to students and the resulting drop in enrollments in computer science programs. Course enrollments are dropping at both the secondary and post-secondary levels (Taulbee, 2003), and the number of young women and minority students studying computing is at an all-time low (The College Board, 2005). The Computer Science Teachers Association (CSTA, 2005a) has identified a number of factors that contribute to this situation, including:

- the lack of a national high school curriculum for computer science education,
- the chronic under-funding of computer science programs in high schools,
- the absence of standards for certification of computer science teachers,
- a shortage of professional development opportunities that would allow teachers to develop and keep their technical and pedagogical skills current,
- the inability of school districts to attract or maintain highly qualified teachers in the face of salary and benefit competition from industry, and
- the lack of understanding on the part of students, parents, guidance counselors, and teachers about computer science in general, how it differs from other areas of computer study, and its newly developing career opportunities.

1.2.1 The Curriculum

In recent decades, computers have come to occupy a pivotal place in our work, personal, and home lives. Accordingly, the field of computer science has expanded to encompass the technical, innovative, conceptual, and psychological ramifications of the presence of the computer at work and at home. As Tucker et al. (2003) noted, however, very little attention has been paid to meeting the need for a well-structured, conceptually based high school computer science curriculum at the time when most students are likely to be building the conceptual frameworks of their intellectual development and exploring the options for their life's work.

To date, efforts to standardize the study of computer science in U.S. high schools have been idiosyncratic and uncoordinated. While other countries have designed and implemented national computer science education programs in order to better prepare their students for the increasingly competitive global economy, attempts to bring coherent computer science education into U.S. high schools failed to address the need to instill a fluent understanding of algorithmic thinking and problem solving, and to provide exposure to software development using programming skills in U.S. students. While various researchers, institutions, and organizations have attempted to define and disseminate new computer science curricula, these efforts have been typified by:

- a persistent shortage of support resources,
- a lack of initial and sustaining funding,
- an absence of on-going, classroom-relevant teacher training, and
- the relentless "projectitis" (Fullan, 2002) that prevents any one educational initiative from receiving anyone's attention for very long.

In addition, the extreme localization of educational (especially curriculum) policy and decision making (which is often relegated to individual schools) and a complete lack of committed engagement by the federal government (despite the growing awareness of the impending skills shortage crisis) have resulted in an abject failure to grapple with this pressing educational issue.

1.2.2 Students and Courses

In 2004, CSTA (2005) surveyed 14,000 high school teachers who defined themselves as computer science, computer programming, or AP computer science teachers. The study's purpose was to create a snapshot of the current state of computer education in the United States, and it revealed some disturbing trends. Despite the growing importance of computer knowledge and skills, only 26% of responding schools required students to take a computer science course, and only 40% of responding schools even offered an Advanced Placement (AP) computer science course. Among students who took an introductory level computer science course, on average only 32% were female. Among students taking the AP computer science course, the number of females dropped to 23%.

CSTA's research is further supported by The College Board (2005), which reported that from 2002–2004 for example, while AP exam taking in other disciplines rose overall by 19%, the number of students taking the computer science A exam dropped by 8%, and the number taking the computer science AB exam decreased by 19%. In addition, in 2004 while 56% of the AP test takers overall were female, among CS AB test takers, only 11% were female (a drop from the 1999 data of 17%), and only 6% were from under-represented minorities.

When asked to explain these increasingly problematic enrollment results, teachers noted that the greatest impediment to students taking high school computer science courses was not the perceived difficulty of the subject matter or even the perception that computer science was a "geeky" course, but rather the lack of time in the students' schedules. As the number of mandated courses high school students are required to take and competition for acceptance to prestigious university programs increase, students have fewer and fewer opportunities to study

computer science because these courses are elective rather than mandatory courses.

Considerable anecdotal evidence also suggests that perceptions concerning career opportunities are playing a role in diminishing student interest in computer science courses. Teachers are reporting that an increasing number of students and their parents believe (based upon media stories rather than marketplace realities) that computer science is no longer a source of rewarding and varied career opportunities. In the absence of solid educational and career information, students are therefore being actively dissuaded from computer science courses and from considering careers in the high tech industry.

1.2.3 The Teachers

Students are also not the only ones affected by the current crisis in computer science education. A 2002 study of 4,000 U.S. high school computer science teachers conducted by the Association for Computing Machinery (ACM, 2005) revealed that 89% of high school computer science teachers indicated that they experience a sense of isolation and a lack of collegial support within schools and school districts. Noting that rapid changes to both technology and teaching provide significant challenges, the teachers also indicated that their greatest professional development need was actually finding time for their own on-going learning (CSTA, 2005). They also indicated that the on-going battle for adequate resources, the lack of acceptance and understanding of computer science as a scientific discipline distinct from technology training, and increasing budget cuts in these times of fiscal restraint deterred many interested and qualified teachers from teaching computer science.

One additional persistent and often ignored problem is that there is little motivation for those with the requisite skills to pursue a career teaching high school computer science. In most jurisdictions, teachers' salaries are so low and the working conditions so unpleasant when compared to other career fields, that it is impossible for education to attract individuals with the appropriate skills. Even for those who are considering a second career in computer science education and for whom salary issues may not be a primary factor, the lack of consistent and readily available information concerning certification requirements make it almost impossible to determine how one should go about preparing for such a career change.

1.3 IS REAL CHANGE POSSIBLE?

Education is a highly complex and multifaceted environment within which an almost unimaginable number of policies, methodologies, strategies, ideologies, and bureaucracies interact (Bauch & Goldring, 2000). And while at its most basic level, the transition of knowledge might appear as a simple process, effective teaching depends significantly upon the contexts within which teachers work—department and social organization and culture, professional associations and networks, community educational values and norms, and secondary and higher education policies (McLaughlin, 1992). These contexts determine not only what is taught and how it is taught, but also the value placed upon a given academic discipline's place within the larger school curriculum and the resources that are made available to instructors.

At present, many of the changes that occur within education are the result of small-scale bottom-up innovation at the individual school level. Usually, such change is structural and superficial and rarely translates into large-scale innovation at the system or national levels (Ballantyne, McLean, & Macpherson, 2003). In effect, such changes tinker with, but do not transform, the educational culture. They do not lead to lasting change because they do not change what people in the organization value and how they work to accomplish it (Fullan, 2002).

To date many initiatives have been launched and much money has been spent on proposed innovations that have resulted in very little change over time, primarily because these initiatives have

failed to grapple with the multiplicity of demands and expectations placed upon schools and teachers. The reasons such undertakings fail are multitudinous, but such failures most commonly result from some or all of the following factors.

1. **Too many messages.** The education system is bombarded with demands for changes from multiple external stakeholders in all curriculum areas, resulting in too much competition for attention and resources. Everyone has an answer, but no one really sees his/her own suggestions within the holistic framework of education.
2. **Poor communication of Return on Investment (ROI).** Outside interests urge and expect schools to change but fail to consider or effectively communicate what the ROI will be for students, teachers, and administrators.
3. **The barrier of vocabulary.** The field of education has its own language where seemingly simple phrases (such as assessment-based learning) can be a minefield for unsuspecting outsiders.
4. **A lack of respect for K–12 educators.** Pre-college educators are often treated with disdain by their university colleagues, by industry, and by politicians. Even those most interested in building partnerships with K–12 frequently undermine their own good intentions with their profound lack of understanding about what it means to teach in a K–12 environment.
5. **Frustration with past failures.** Over the years, industry representatives have become increasingly frustrated by the frequency with which they have contributed to initiatives with little or no lasting impact or return. The problem is that those promising to ensure these changes do not have a foundational understanding of issues of change or the efficacy of change agents in the K–12 arena.

Systemic educational change demands a profound understanding of and engagement with the culture in which the change must take place. While achieving the level of change required to address the current issues in high school computer science education in U.S. schools may seem like an impossible challenge, time has shown that when sufficient recognition, resources, and long-term commitment are brought to bear on a national challenge, the unthinkable can be achieved.

1.4 HOW CAN THIS DOCUMENT HELP?

The purpose of this document is to set forth some of the important factors that must be considered in order to achieve a substantial improvement to high school computer science education, an improvement that must occur if we are to meet the shifting knowledge needs of our citizens and so remain a competitive force within the global economic community.

To achieve this, we must begin by acknowledging that other nations may have knowledge they can share with us, that will help us ensure that we use our resources wisely and do not repeat mistakes that others have already made. The second chapter of this document, therefore, provides a comprehensive review of the international body of research relating to high school computer science education. It examines the issues that have been raised, the challenges that have been identified, and the solutions that have been proposed.

The third chapter provides an analysis of five countries, including the United States, in which efforts to design and implement a comprehensive high school computer science curriculum have been undertaken. This chapter demonstrates that defining the curriculum is really just a first step and that the factors that determine the extent to which that curriculum is actually implemented may in fact be more important than the curriculum itself.

The last section draws from all of the previous sections of the document to provide a comprehensive set of recommendations and strategies that will significantly increase the likelihood that our national efforts to improve high school computer science education will achieve long-term success.

1.5 REFERENCES

ACM K–12 Task Force. (2005). *ACM K–12 Task Force Curriculum Survey Results.* Retrieved December 22, 2005, from http://csta.acm.org/Research/sub/TuckerSurveyResults.html

Ballantyne, R., McLean, S. V., & Macpherson, I. (2003). *Knowledge and skills required for creating a culture of innovation: Supporting innovative teaching and learning practices.* Brisbane: Faculty of Education, Queensland University of Technology.

Bauch, P. A., & Goldring, E. B. (2000). Teacher work context and parent involvement in urban high schools of choice. *Educational Research and Evaluation, 6*(1), 1–23.

Computer Science Teachers Association (2005a). *Strategic Plan.* Retrieved December 22, 2005, from http://csta.acm.org/About/sub/CSTAStrategicPlan.html/.

Computer Science Teachers Association. (2005). *Results of the national secondary computer science survey*. Retrieved October 16, 2005 from http://csta.acm.org/Research/sub/CSTANationalSurvey2004.html

Fullan, M. (2002). The change leader. *Educational Leadership, 59*(8), 16–21.

Information Technology Association of America. (2002). *Bouncing Back: Job skills and the continuing demand for IT workers.* Arlington, VA: Information Technology Association of America.

McLaughlin, M. W. (1992). How district communities do and do not foster teacher pride. *Educational Leadership, 50*(1), 33–35.

National Research Council Committee on Information Technology Literacy. (1999). *Being fluent with information technology.* Washington, DC: National Academy Press.

Sargent, J. (2004). An overview of past and projected employment changes in the professional IT occupations. *Computing Research News, 16*(3), 1–22.

Shackelford, R. (2005). *Why can't smart people figure out what to do about computing education.* Presentation at the CSTA/ISTE Computer Science and Information Technology Symposium, Philadelphia, PA.

Taulbee, O. E. (2003). *2002–2003 Taulbee study: Undergraduate enrollments drop; Department growth expectations moderate.* Retrieved December 22, 2005, from http://www.cra.org/statistics/

The College Board. (2005). *AP report to the nation.* Retrieved December 22, 2005, from http://www.collegeboard.com/about/news_info/ap/2005/index.html

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C. and Verno, A. (2003). *A model curriculum for K–12 computer science: Report of the ACM K–12 Education Task Force Computer Science Curriculum Committee.* New York, NY: Association for Computing Machinery.

CHAPTER TWO

CHALLENGES AND PERSPECTIVES IN HIGH SCHOOL COMPUTER SCIENCE EDUCATION: A RETROSOPECTIVE LITERATURE REVIEW

2.0 INTRODUCTION

Computer science is considered a young and a rapidly developing discipline, and its "extremely short history has led to a diversity of opinions about its very substance" (Gal-Ezer & Harel, 1998, p. 79). While many educators believe that computer science should be taught to prospective students as an independent subject (Gurbiel, Hardt-Olejniczak, & Kolczyk, 2005) and as a full-fledged scientific subject on a par with other scientific subjects (Gal-Ezer, Beeri, Harel, & Yehudai, 1995), others contend that it should be offered as a set of integrated courses that link computing to other subjects (Dagienë, 2005). This discussion about how computer science should be taught is further complicated by the fact that many people have difficulty identifying exactly what computer science is. As a result, educators have been deliberating how to portray the field to others (especially to newcomers) in a compact, compelling, and coherent way (Denning, 2004), and how to expose others to a bird's-eye view of the field (Gal-Ezer & Harel, 1998).

In recent years computer science education research has received increasing interest and recognition in the computer science education community (Holmboe, McIver, & George, 2001; Dale, 2002; Almstrum, Hazzan, & Ginat, 2004; Fincher & Petre, 2004; Goldweber, Fincher, Clark, & Pears, 2004). Holmboe, McIver, & George (2001), for example, provided the following general categorization of computer science education research to date:

- New, untested ideas and methods for teaching and learning.
- Reports from the trenches—sharing ideas and techniques for teaching in particular courses.
- Discussion of theory—with regard to epistemological theories such as constructivism.
- Computer-aided learning and intelligent systems—aimed at learning about the students' ways of thinking and giving feedback.
- Expert/novice differences as a means of setting up benchmarks for novice achievement.
- Empirical studies, which focus on real programming, as a basis for creating effective tools for teaching programming.

Despite the increased interest in computer science education research by the educational research community, there are still many open questions relating to computer science education at the high school level for which answers are still very much needed. These include:

- Should we really teach computer science in high school, or is it best left to be taught at the post-secondary level?
- What are the goals of high school computer science education, which topics should be taught, and how should the curriculum be organized?
- How can we successfully promote a massive implementation of a new computer science curriculum?
- What pedagogic strategies and methods of instruction should be applied to reduce the complexity and instability that characterizes a dynamic discipline?
- How should computer science teachers best be prepared to cope with the increasing complexity and rapid change?

- How should we cope with students' preconceptions and expectations, and how should we better address gender differences and minorities' special needs?

All of these questions have become more pressing in light of current concerns regarding dropping enrollments in computer science courses at all educational levels and predictions of high tech worker shortages worldwide. For many countries, the relationship between an educational system that produces workers capable of building the technology tools that the rest of the world will use and long-term economic prosperity is becoming increasing clear. This has led to efforts to improve computer science education, especially at the high school level, where many students have their first opportunity to test their interest and ability in this field. In the United States, however, the situation is far more tenuous as the lack of national standards in both curriculum and teacher certification have resulted in significant discrepancies in both quality and quantity, not just from state to state but from school to school. In addition, lack of administrator understanding of the nature and importance of this complex and evolving discipline may actually be diminishing support for computer science as part of the high school curriculum at exactly the time that it should be receiving more attention and support.

The aim of this review of computer science education research is to illuminate various aspects of computer science education at the high school level, and to summarize reported studies and field experiences, as well as educators' recommendations for addressing the questions outlined above.

2.1 COMPUTER SCIENCE EDUCATION IN HIGH SCHOOLS

The research on high school computer science education is complex, not just because there is a lot of it and it focuses on many different aspects of education, but also because it is truly international in scope, with researchers from many countries contributing to the growing body of knowledge. This section provides a brief overview of high school computer science education in the United States, Israel, and European countries with the goal of providing a snapshot of its current focus and status internationally.

2.1.1 United States

In fall 2004, the Computer Science Teachers Association (CSTA, 2005) surveyed 14,000 high school computer science, computer programming, and computer applications teachers in an attempt to provide a baseline understanding of high school computer science education in the United States. The survey respondents (1,047 teachers) provided a rich body of data on the current state of computer science education including the following (Roberts & Halopoff, 2005).

- Many institutions teach computing courses at a lower level than the Advanced Placement (AP) curriculum suggests; programming is the most commonly covered topic, followed by hardware, ethics, graphics, and web development (in that order). Courses that focus on applications are less likely to cover programming and vice versa.
- Schools offer more pre-AP computer science courses than AP courses, and the AP courses tend to have significantly fewer students.
- The percentage of women declines from 32% in the pre-AP courses to 23% in the AP program.
- Teachers believe the primary reason that students are not taking computer science courses is that they cannot fit them into their tightly prescribed timetables.
- Teachers perceive the "rapidly changing technology" as a serious problem and the greatest challenge in teaching computer science.
- Teachers indicated "time for training" as an important professional development need and preferred short workshops as the most effective

method for delivering professional development.
- Teachers are surprisingly poorly informed about the rules concerning curriculum content and teacher certification and the administrative structures within their own states.

2.1.2 Israel

In Israel, computer science has existed as an autonomous subject in the high school curriculum since the mid-1970s. As in many countries, however, despite its successful implementation and the involvement of many highly respected educators and researchers in the development and evolution of its curriculum, it still suffers from the perception that it is not a full-fledged scientific subject like physics, biology, or chemistry.

Originally, the curriculum focused on programming and included elective Information Technologies modules such as electronic spreadsheet language. Since 1991, however, a new curriculum that places significantly more emphasis on principles and theoretical aspects has been gradually implemented in Israeli high schools. The new curriculum "emphasizes the foundations of algorithmic thinking, and teaches programming as a way to get the computer to carry out an algorithm" (Gal-Ezer, Beeri, Harel & Yehudai, 1995, p. 73). The program is modular and comes in two versions—basic and advanced. It includes mandatory Fundamentals and Software Design modules as well as elective modules, including Second Paradigm, Theory, and Applications.

The success of the high school computer science curriculum in Israel is largely due to the care with which the government planned the implementation process and the resources that were put in place to support that implementation. This support included the development of course materials (learning materials for students and corresponding teacher guides) and an intensive in-service teacher training program (Gal-Ezer, et al. 1995; Haberman & Ginat, 1999).

2.1.3 Europe

A substantial body of research on computing education in Europe has supported the argument that educational standards, continuous teacher training, and enthusiastic teachers who permanently engage themselves in new tools and concepts are critical factors for the successful establishment and maintenance of high school computer science education, more commonly called informatics, in European schools (Reiter, 2005; Micheuz, 2005; Dagienë, 2005; Dorninger, 2005; Kuznetsov & Beshenkov, 2005). According to Mittermeir (2005), however,

> the penetration of personal computing and the (almost) ubiquitous presence of certain types of application software have had substantial impact on computer science, more commonly called informatics, education across Europe. Over time, the principles of abstraction and algorithmization gave way to intellectually less rewarding topics. (p. 2)

As a result, a number of countries have drifted toward a more Information Technology (IT) focus that was less rigorous and less embedded in science.

In 2004, a study of twenty European Economic Union member states as well as Bulgaria, Iceland, Liechtenstein, Norway, and Romania (Eurydice, 2004) revealed that studies of information and communication technology form a part of the compulsory curriculum in the upper secondary level in all countries. The ways in which that curriculum is implemented, however, differ considerably from country to country. In the majority of European countries, information and communication technology (or informatics) is included in the national core curriculum as an independent subject. In Ireland, however, students do not study programming in secondary schools (Bergin & Reilly, 2005), and in Finland and Italy, information and communication technology is no longer taught as a separate subject but rather as a tool to support learning in other curriculum areas (Kavander & Salkoski, 2004; Grandell, 2005; Cartelli, 2002).

In Poland, all students study informatics in primary schools and in middle school. High school students also take a mandatory *Information Technology* course that included topics such as networks and

multimedia tools for managing information. An elective course called *Informatics* is also available for students who are interested in computer science as an element of their future education. This course focuses on the science of computing with the goal that students understand how "computer science as a discipline is connected with designing and implementing new systems of information processing" (Gurbiel, Hardt-Olejniczak, & Kolczyk, 2005, p. 49).

Informatics has been part of the high school curriculum in Austria since 1985 when it was first introduced for all students in Grade 9. The recent reform of the upper secondary level of the academic schools also added an elective *Informatics* course for students in Grades 10–12 that reinforced integration of informatics methods in other subjects as well (Micheuz, 2005). The development of informatics in the Austrian schools is characterized by constant changes and improvements in hardware, software, and by corresponding pedagogical approaches as well. "Didactics in Informatics changed in many schools from a programming and algorithm orientation to an application oriented approach" (Micheuz, 2005, p. 27).

Informatics as a separate subject was also established in Lithuanian schools in the late 1970s and early 1980s. The subject is compulsory for students in Grades 9–12. Its main goal is to develop students' information culture in a broad sense. Students in Grades 11 and 12 may also choose an optional advanced course that focuses on programming, databases, and multimedia. "During the lessons the integrative nature of the course is being stressed; students are prompted to see parallels with other subjects" (Dagienë, 2005, p. 56).

2.2 WHO ARE WE, AND HOW SHOULD WE PORTRAY OURSELVES ?

2.2.1 An Inconsistent Self-image

As is apparent from the very brief description of high school computer science education in a number of countries, the discipline suffers from something of a crisis of definition. As Gal-Ezer and Harel (1998) and Holmboe et al. (2001) have noted, even among computer scientists, there is no clear agreement on the name of the field (*computer science, informatics, information science*), and efforts to determine its exact contents have been similarly challenging. Researchers have identified many factors to account for this lack of consistency. Gal-Ezer and Harel (1998) posited that the disagreements regarding the essence of computer science are due to a dichotomy between the mathematical and engineering facets of the field and to dichotomies within each facet. Denning, Comer, Gries, Mulder, Tucker, Turner, and Young (1989) similarly described computing as sitting "at the crossroads among the central process of applied mathematics, science and engineering" (p. 11).

In addition, constant changes both in the technology and in our understanding of computing as a whole have made it extremely difficult to create a static representation of the core elements of the discipline. In 1989, when the number of core technologies was much smaller, Denning et al. (1989) suggested that an intellectual framework for the discipline based upon a matrix model of nine core technologies should replace the common definition of computer science in terms of its three major paradigms: theory, abstraction, and design. More recently, however, in an effort to more accurately portray the rapidly developing and now more complex field, Denning (2004) recommended that a new framework of *great principles* and computing practices was required to replace the conception of the discipline as a set of core technologies.

2.2.2 An Incorrect Public Image

Computer science's public image, like its self-image, is highly problematic. Research has indicated that beyond disagreements about the nature and content of the discipline, computer science is plagued by a number of powerful misconceptions that are shared not only by "real" outsiders (members of other disciplines, policy makers, school principals, etc.) but

also by students, prospective teachers, in-service teachers, and others involved in computer science education. Here is a series of brief descriptions of the most common of these misconceptions.

2.2.2.1 *Computer Science Equals Programming*

The most common misconception about computer science arises from the fact that it is still widely perceived as a programmer's field (Denning, 2004). Denning et al. (1989) stated that the view that *computer science equals programming* is especially strong in most of the curricula because introductory courses focus (sometimes exclusively) on programming and this focus limits the ability to reliably describe the intellectual substance of the discipline. Greening (1998) and Gries (2002) also demonstrated that students' most common misconceptions about the discipline are compatible with the *computer science equals programming* view. Schollmeyer (1996) further noted that this limited view of the discipline could have a profoundly negative effect on students' expectations of further computer science studies in college.

2.2.2.2 *Computer Science Equals Computer Literacy*

The tremendous effort undertaken to integrate computer use across the curriculum in K–12 education has led many educators and administrators to confuse computer science education with computer literacy, and as a result, in many schools today there is no adequate separation between teaching computer science, computer literacy, and using computers to teach other subjects (Gal-Ezer & Harel, 1998; Grandell, 2005). Mittermeir's (2005) study of national perspectives on teaching Informatics during the last twenty years in secondary schools in Europe concluded that intellectually challenging curriculum content (for example, abstraction and algorithmic thinking) is often sacrificed to the teaching of simple software applications such as word processors and spreadsheets.

Sabin, Higgs, Riabov, and Moreira (2005) also criticized high school computing courses in the United States because they are sharply polarized between two curricular approaches: (1) basic training in using Microsoft office applications and (2) strong emphasis on AP programming (as preparation for AP exams). Their concern was that neither of these approaches reliably portray the essence of computer science, and they both reinforce the *computer science equals computer literacy* and the *computer science equals programming* misconceptions. Sabin et al. (2005) concluded that high school students should be exposed to an extracurricular enrichment computer science program that focuses on fundamental concepts in computing and some of its relevant applications and interweaves attractive learning activities.

2.2.2.3 *Computer Science is a Tool for Studies in Other Disciplines*

In many countries there has been considerable confusion between teaching computer science using a real-world approach and simply using the tools of computer science to support education in other (particularly scientific) disciplines. Today there is a large body of research evidence supporting use of computer science principles and problem-solving methods to solve "realistic" problems in various domains (Sims-Knight & Upchurch, 1993; Dagienë, 2005). Stevenson (1993) for example, argued that students need to acquire a scientific-engineering interdisciplinary approach to solve complex problems in various domains; hence, computer science should not be taught isolated from mathematics and other sciences. Scherz and Haberman (2005) also recommended that students should use the tools of computer science to develop knowledge-based projects related to descriptive topics of a qualitative nature.

While these calls for an interdisciplinary approach were never intended to be construed as an argument for eliminating computer science as a distinct subject of academic study, in some countries, such as Finland

(Kavander & Salakoski, 2004; Grandell, 2005), there has been a profound shift toward teaching computer science only as a tool for addressing problems in other academic disciplines (Gal-Ezer & Harel, 1998). It is ironic to note this misinterpretation of the essence of the interdisciplinary pedagogic approach may have resulted precisely because computer science educators attempted to integrate computers into all high school courses as a means of promoting high school students' choice of computer science as a major. This approach is illustrated by O'Lander (1996) who stated, "This will make more students comfortable and expert at using computers for a wide variety of purposes, and therefore increase the probability that more of them will choose computer science as a major in college" (p. 29).

2.2.2.4 *Computer Science is Not a Scientific Discipline*

Members of the computing community of practice consider computer science as a scientific discipline (Denning, 2004). For example, one of the underlying principles of the new high school program in computer science in Israel is "Computer science is a full-fledged scientific subject. It should be taught in high school on a par with other scientific subjects" (Gal-Ezer et al., 1995, p. 75). Computer science courses in high schools, however, seem to generally have a second-class status when compared to other science courses such as biology, physics, and chemistry, and rarely count towards satisfying the science requirements in the high school curricula (Morris & Lee, 2004; Tucker, Deek, Jones, McCowan, Stephenson, & Verno, 2004). In order to establish a correct public image, Denning (2004) suggested that computer science should portray itself in the same manner that the mature sciences (e.g., physics, biology, and astronomy) portray themselves, namely with a principle-based approach that "promotes understanding from the beginning and shows how the science transcends particular technologies" (p. 337). Specifically, he recommended that computer science, like other sciences, use a set of interwoven stories about the structure and the behavior of the field elements.

2.2.2.5 *Computer Science is a Male Field*

Both male and female students continue to see computer science as a primarily male field and to make their career choices accordingly. As Moorman and Johnson (2003) note, "Women in computing still perceive themselves as strangers in a strange, male-dominated land" (p. 193). Educators believe that the lack of interest in computer science among girls might be rooted in stereotypes of computer science formed early in their school experience. Girls think that computer science is "a boring subject, devoid of interesting applications and stimulating only to 'geeks'." (Graham & Latulipe, 2003, p. 322). Consequently, female enrollment in undergraduate computer science programs has been constantly declining, and women tend to be underrepresented in computing courses (Stiller & LeBlanc, 2003; Graham & Latulipe, 2003).

2.2.3 Challenging Student Preconceptions

Students might actually possess the misconceptions presented above even before they begin to learn the subject in school. Rountree, Vilner, Wilson, & Boyle (2004) argue that the youngsters' increased access to home computers may cause many students to form a misleading view of what computer science is. Many potential students also incorrectly believe that most software professionals will spend a significant amount of their career programming games (Latulipe & Graham, 2005). Furthermore,

> Young people who like playing computers and surfing the Internet declare their interest in opting for an elective subject [i.e., computer science]. They do not know what it is really about, because there was no signal … that problem solving,

> algorithms, and programming need some skills of thinking, reasoning, and understanding mathematical concepts (Gurbiel, Hardt-Olejniczak, & Kolczyk, 2005, p. 51).

Numerous researchers have noted that common misconceptions about computer science "raise obstacles in encouraging students to pursue computer science majors, minor in computer science, or simply choose computer science electives in their program of studies" (Sabin et al., 2005, p. 177). Gupta and Houtz (2000), for example, argued that students may hesitate to enroll in computer science programs because of the negative perceptions about technology careers and a lack of understanding of the skills needed to succeed in these careers. Foley (2004) also supported this contention, noting that since programming is often considered as a "solitary activity practiced by so-called geeks," students might avoid studying computer science. The general conclusion of the body of research is that educators must be aware of common misleading preconceptions and expectations and make every effort to deal with prospective students and find ways to motivate them to pursue computer science programs (Sabin et al., 2005).

In his study of university students' beliefs about computer science, Greening (1998) attempted to determine whether student misconceptions were related to high early student dropout rates in first-year computer science courses or to students deciding not to enroll in computer science courses in the first place. Greening's study revealed the following.

Most students were unable to give an approximate definition of computer science or offered poor or erroneous descriptions of the domain.

Many potential students referred to computer science in terms of the study of the computer as a machine and indicated that computer science is about the use of computers for running applications.

Most of the students shared the perception that first-year computing courses should teach programming, general usage skills, and provide familiarity with general computing skills.

A significant percentage of potential students expected to learn about computers and society-centered issues.

These results led Greening to conclude that students did not understand the discipline of computer science or its goals, and that effort required to address student misconceptions should be "directed at potential students in order to educate them about the actual nature of computer science education, and directed at computer science education in order to acknowledge changing expectations of potential students" (p. 154).

Researchers have also attempted to identify the factors affecting high school students' choice of computer science courses, as well as the factors leading to the decline of computer science majors. For example, O'Lander's (1996) study of high school students who take computer science courses focused on students' attitudes and demographic and background characteristics. The results of this study clearly indicated gender differences in favor of the boys. As a result, O'Lander recommended that following changes be made to ensure that computer science be perceived as more attractive to all students:

- a unified computing curriculum should be established to eliminate disparities,
- computing teachers' certification should become a requirement (ensuring that competent teachers implement a challenging, interesting curriculum, and eliminating or minimizing the current practice of teachers certified in other fields teaching computer science courses in high school, and
- computers should be integrated into all high school courses.

Gupta and Houtz (2000) conducted a similar study of high school students. The primary purpose of their study, however, was to better understand the perceptions and attitudes of females and minorities towards computer use and technology careers. The students in Gupta and Houtz's study identified keyboarding as the primary skill necessary to succeed in IT careers, followed by computer skills,

programming, and math (in that order). This study also showed that boys were significantly more interested in enrolling in Information Technology programs and pursuing IT careers than girls were. The researchers therefore recommended that schools:

- develop a high-standard, uniform, mandatory high school computer science curriculum;
- use software that appeals to both genders;
- make computers exciting and challenging for students;
- implement a vigorous computer career counseling program; and
- explore race differences and interest in computer careers.

2.3 HIGH SCHOOL COMPUTING STUDIES AND ACADEMIC SUCCESS

Over the years, many professional bodies have called for the creation of a standardized computer science curriculum for high school students as an important tool for preparing students for post-secondary studies. The report of the *ACM Task Force on the Core of Computer Science*, which laid the foundation for the computer science curriculum for the major colleges, for example, recommended that students enter computer science degree programs with some programming skills, presumably learned in high school (Denning, Comer, Gries, Mulder, Tucker, Turner, & Young, 1988). Yet despite these organizational recommendations, some computer science professors have long maintained that studying computer science in high school is in fact detrimental to students (Taylor & Mounfield, 1989). Research, however, largely disproves this contention.

In general, the research findings indicate that taking computer studies in high school is a positive factor for readiness for the college curriculum and college success (Franklin, 1987; Ramberg, 1986; Taylor & Mounfield, 1989). Specifically, researchers found evidence that prior exposure to computers, whether in high school or college programming courses (Ramberg, 1986; Taylor & Mounfield, 1989; Cantwell-Wilson & Shrock, 2001; Cantwell-Wilson, 2002), or even in computer literacy courses (Ramberg, 1986), is a critical factor for success in academic computer science studies. Moreover, a "prior programming course builds a foundation of logic background" (Ramberg, 1986, p. 37). Gal-Ezer and Harel (1998) have noted, however, that it is important that students' programming experience be of a high quality, and cover not only coding but also the design of algorithms (underlying programs), and basic considerations of correctness and efficiency.

Campbell and McCabe (1984) also attempted to determine the specific factors that influence success in the first year of a computer science major. Their study indicated that students' background in high school mathematics and science is one important factor for success. Based upon these results, they concluded that student participation in high school courses that focus on problem solving and scientific reasoning may contribute to the success of freshmen with a computer science major.

These conclusions were further supported by Franklin (1987). Franklin's study aimed at determining whether high school computer science courses were a predictor of success in college entry-level courses. The results showed a highly significant relationship between success in the entry-level college course and the completion of one or more high school computer science courses. A more recent study, conducted by the Open University of Israel, also revealed that students with previous programming experience were more successful in a university introductory level computer science course than those who reported having no previous programming experience (Gal-Ezer, Vilner, & Zur, 2003; Rountree et al., 2004). Sabin et al. (2005) concluded that an early start in studying computer science is appropriate in high school for those students whose professional future is related to computer science and emphasized the importance of computing courses designed to motivate prospective students to pursue computer science programs.

It is important to note, however, that most of the

research literature on predictors of success in computer science studies refers to traditional (imperative-first) introductory courses. Recently, as a result of the trend towards teaching introductory computer science courses using an objects-first approach, researchers are being challenged to investigate the relationship of the objects-first approach to the previously identified predictors for success in imperative-first introductory courses. Two such studies were conducted with contradictory results. A study aimed at assessing the effects of prior programming experience on the success in an introductory objects-first course revealed that prior programming experience (not general experience or experience with object-oriented programming languages) had no significant influence on success. Hence, the researchers concluded that prior programming experience is not a predictor of success for the objects-first introductory course (Ventura & Ramamurthy, 2004). In contrast, Hagan and Markham (2000) found that students who had experience in at least one programming language at the beginning of an introductory Java programming course performed significantly better than those with none, and that the more languages with which a student has experience, the better her/his performance tends to be. Furthermore, the style of the programming language previously learned was not shown to be significant.

2.4 WHAT SHOULD WE TEACH AND HOW?

The main goals of high school computer science education are to ensure that students are comfortable with everyday computing activities, to open a window of opportunity for young students in the field of computing, to educate them about the nature of the field, and to motivate students who have the ability and interest to continue their academic studies in this discipline. The aim is to expose the students to a fundamental scientific domain whose principles are characteristic of algorithmic thinking as well as system-level perception (Denning et al., 1989; Gal-Ezer et al., 1995).

The rapid growth of the field of computing and the continuous emergence of new computing technologies requires educators to give considerable thought to what and how they should teach. It requires them to provide a coherent view of computer science in a comprehensive and appealing way and to consider students' perceptions and expectations and any factors that may affect students' further studies in the field. In addition, high school computer science educators must determine what topics are required to best prepare students for college-level computer science courses (Schollmeyer, 1996), and how the long-term needs of students can be addressed so that they will aspire to become part of the dynamic technologies-based world (Pham, 1997; Mitchell, 2002).

Educators and scientists also argue that knowledge of fundamental concepts of computer science is essential for understanding the technology that facilitates the "real world" and that these fundamental concepts should be the core of a high school curriculum. For this reason, it is important to organize the curriculum with a suitable balance between conceptual, experimental, and practical issues (Merritt, 1995; Gal-Ezer et al., 1995). As a scientific discipline, computer science has lasting and fundamental values (*essential* characteristics), which are independent of computing technology's change (*accidental* characteristics). Hence, teaching computer science should emphasize the scientific facets of the field, along with the knowledge and skills that are independent of specific computers and programming languages (Gal-Ezer et al., 1995). The current approaches to high school computer science education, however, often hide the vision and the grand challenges of computer science from students because they focus exclusively on developing computer programming skills and ignore the basic precepts of the scientific method that form the foundation in natural sciences (Lee, 2004).

Also, computing is a dynamic field, and the discipline of computer science has continued to evolve

since its inception. This has presented educators with two additional pedagogical challenges:

- **complexity**—an increase in the programming details students must master, and
- **instability**—rapid changes in the programming languages and tools.

Distinguishing between and separating *essential* and *accidental* characteristics of the subject being learned would help eliminate this *complexity* and *instability*. In an effort to directly address both the complexity and the instability of the software, for example, Roberts (2004) argued for the development of a simple and stable environment explicitly designed for pedagogical use. Specifically, he concluded that programming language instruction should emphasize essential principles and conceptual aspects of the particular programming paradigm while decreasing the complexity of accidental implementation features that are technology-based.

Denning (2004) noted the need to also address these issues at a curriculum level. He suggested a framework for organizing a computing curriculum around great principles and practices that "promote a greater understanding of the science and engineering behind information technology" (p. 340). This framework offered a balance between concepts and practice. It also provided a stable context for the core technologies of computing related to both mechanics and practices.

In ACM's *Model Curriculum for K–12 Computer Science,* however, Tucker et al. (2003) outlined a comprehensive model curriculum for K–12 that builds upon the notion of IT fluency put forward by the National Research Council (1999). In this context (education), IT fluency is defined not only as the capability to use today's technology (the typical definition of IT literacy) but also the capability to independently learn and use new technology as it evolves. In this way, IT fluency expands the conception of a successfully educated person to include the active use of algorithmic thinking to solve problems. The ACM *Model Curriculum* is a comprehensive four-level model for K–12 computer science education that focuses on fundamental concepts. It incorporates and enhances the National Educational Technology Standards (International Society for Technology in Education, 2002) for grades K–8 (*Foundations of Computer Science*) and then proposes a set of three courses:

- *Computer Science in the Modern World,*
- *Computer Science as Analysis and Design,* and
- a special *Topics in Computer Science* course that could include AP courses and courses leading to industry certification.

2.4.1 Guiding Principles

Many researchers and educators have attempted to define the underlying principles or concepts that are foundational to all of high school computer science. This section provides a brief description of the most commonly agreed-upon principles.

- **General capabilities and skills**—Students should develop a wide range of cognitive capabilities and practical skills, independent of specific technologies, to enable them to continue to learn and adapt quickly to new environments and work practices (Pham, 1997; Gal-Ezer et al., 1995). They must also acquire a breadth of skills required for future employment such as writing, communication, and presentation, and other skills that enable them to continue to learn and adapt quickly to new environments and work practices (Pham, 1997; Mitchell, 2002).
- **Breadth-first approach**—Students should acquire a broad overview of the field to construct a comprehensive picture of computer science as a discipline. They should have a sense of the history of the discipline so that they can better appreciate how computers have been used to address real problems (Impagliazzo & Lee, 2004; Hazzan, Impagliazzo, Lister, & Schocken, 2005). Different programming paradigms should be taught to acquire alternative ways of algorithmic thinking as well as different problem-solving methods (Vandenberg & Wollowski, 2000; Gal-Ezer et al., 1995).

- **Understanding the interplay between theory and practice**—Students must understand not only the theoretical underpinnings of the discipline but also how that theory influences practice. Accordingly, conceptual and experimental issues should be interwoven throughout the program (Gal-Ezer et al., 1995; Ben-Ari, 2002). However, since beginning computer science students do not have an effective model of the computer, "the seductive reality of the computer must not be allowed to replace construction of models" (Ben-Ari, 2001); meaning that programming exercises should therefore be delayed until class discussion has enabled students to construct a good model of the computer.
- **Problem solving and algorithmic thinking**—"Students must be taught and coached in the algorithmic way of thinking" (Gal-Ezer & Harel, 1998, p. 83). To best prepare high school students for computer science courses in college, the emphasis at the high school level should not be limited to the instruction of the syntax of a programming language. Rather, students' problem-solving skills should be addressed. This includes emphasis on teaching problem-solving methodologies and critical thinking skills (Fadi & McHugh, 2000) as well as program specification and design (Schollmeyer, 1996). Integrated learning environments should also be designed to support novices through all problem-solving and programming stages (Fadi & McHugh, 2001).
- **System-level perspective**—Students should be taught what will be expected from them in the "real world," specifically, what is actually required to write and maintain computer programs and large software systems. They must therefore develop a high-level understanding of systems as a whole: the structure of computer systems and the processes involved in their construction and analysis (Gal-Ezer & Zeldes, 2000; Schollmeyer, 1996). Design principles should be taught explicitly, otherwise, the first programming course might turn into a course on syntax no matter how simple the underlying programming language is (Felleisen, Findler, Flatt, and Krishmamurthi et al., 2002).
- **Integrative and interdisciplinary knowledge**—Denning et al. (1989, p. 13) suggested that "interacting with other disciplines to support their interests in effective use of computing" should be an integral part of a curriculum for computing majors. Stevenson (1993) presented a possible foundation for computational science, which is a much wider domain than computer science, and embodies a scientific-engineering interdisciplinary approach to solving very difficult problems. He remarked that computer science students "can easily see computer science as devoid of meaning and programming devoid of empirical import" (p. 9). He also noted that students are not well prepared to cope with complex problems that involve mathematical, scientific, and engineering reasoning. Hence, the computer science studies in high school should be enriched with mathematics and sciences, with emphasis on numerical analysis applications, and scientific software engineering concepts, and students should be educated to employ an interdisciplinary team approach (Stevenson, 1993).
- **Theoretical knowledge**—Educators believe that it is important to expose high school students to advanced topics of computer science (e.g. computational models, modeling, and parallel computing) to enable them to become familiar with some of the theoretical aspects of computer science (Armoni & Gal-Ezer, 2003; Loidl, Muhlbacher, & Schauer, 2005; Ben-David Kolikant, 2004). These concepts may be introduced in a traditional way; however, they can be made more comprehensible when demonstrated by means of new media such as applets (Loidl et al., 2005).
- **Familiarity with common principles**—students must recognize recurring concepts and principles such as abstraction, complexity, modularity, and reusability (Denning et al., 1989; Gal-Ezer et al., 1995).

- **Comprehensive programming**—Programming should be taught as a means for getting the computer to execute an algorithm. It should be taught in a broad sense, "covering not only the coding act itself, but also the design of the algorithms underlying the programs and, to some extent, considerations of correctness and efficiency" (Gal-Ezer & Harel, 1998, p. 82). High school programming classes should attempt to prepare the students to be successful in corresponding college-level classes by teaching material that is universally applicable to any programming paradigm (Schollmeyer, 1996). Fincher (1999), however, has questioned the argument that students must learn programming first and disputed the notion that there is a distinguished order to concept acquisition. "Instead of accepting the view that students need to learn to code and that from this experience they will learn complex, transferable skills…. leaving the students to abstract these for themselves," (p. 12a4-5) she argued, one should first identify the skills that should be acquired and support student learning to achieve these goals.
- **Meaningful learning of computing principles in attractive experimental environments**—Teachers should motivate students to endure the rigors of traditional computer science programs by enticing students with exciting, accessible, and leading-edge courses. Guzdial and Solloway (2003) and Stiller and LeBlanc (2003), for example, have suggested that introducing computing in an authentic environment, such as the world of audio and visual media, would motivate students and interest them in the rest of computer science.
- **Teaching from an historical perspective**—Böszörmenyi (2005) recommended that in addition to teaching formal definitions and programming practice, teachers should provide students with an understanding of the great ideas of the major thinkers in computer science and the context in which these ideas were developed.
- **Diminishing common computer science misconceptions**—Teaching and learning activities should be designed to treat common misconceptions of the very essence of computer science. In particular, it is important to diminish the computer science equals programming misconception by developing an appropriate fundamentals course that emphasizes the breadth of the discipline beyond the interest in computer programming (Gries, 2002; Gal-Ezer & Harel, 1998; Peckham, DiPippo, Reynolds, Paris, Monte, & Constantinidis, 2000; Vandenberg & Wollowski, 2000).
- **Gender differences and minority issues**—The curriculum should be designed to address the under-representation of women and minority students in computer science programs (Rich, Perry, and Guzdial, 2004; Guzdial & Soloway, 2003; Graham & Latulipe, 2003; Payton, 2003; Stiller & Leblanc, 2003).

2.4.2 Teaching the Key Concepts

There is also a large body of research that examined the importance of taking a conceptual approach to teaching computer science in high schools. In this section, we briefly summarize these research studies with the intention of highlighting the key findings and suggestions for curriculum contents and enhancements.

2.4.2.1 *Basic Computing Concepts*

Several researchers (Perkins, Schwartz, & Simmons, 1988; Sleeman, Putnam, Baxter, & Kuspa, 1989; du Boulay, O'Shea, & Monk, 1989) have noted that students lack an adequate knowledge of programming and also demonstrate many misconceptions about basic computing concepts because they lack viable mental models of the computer. Ben-Ari (2001) argued that the common difficulties experienced by novices could be explained by computer science education being heavily weighted on the side of bricolage and

involving too much and/or too early use of the computer (e.g., try it and see what happens, endless debugging, and avoiding abstraction). As a result, he concluded, high school students aspire to program directly on the computer, and they sometimes complain that algorithm development and analysis, instead of just getting on with writing and debugging programs, is a waste of time.

2.4.2.2 *Algorithms*

A number of researchers have addressed the special needs of novice programmers. Haberman and Ben-David Kolikant (2001) concluded that teaching novice students basic concepts in computation is not a simple task. In fact, they reported that educators may be inclined to believe that these concepts are trivial and therefore easy for students to understand, when in fact, novice programming students may not understand them at all. They therefore recommend that teachers carefully choose an instructional method that clearly demonstrates and reinforces even seemingly simple concepts.

Haberman, Averbuch, and Ginat (2005) also concluded that novices tend to have limited understanding of algorithms, and so often consider an algorithm as a stand-alone product, thus ignoring issues such as correctness and efficiency. Moreover, they concluded that students perceive algorithms as operational process emulation elements, rather than structural concepts that yield a specified I/O relationship. As a result of these findings, the researchers suggested that students should be taught that explanations and justifications are inherent elements in the solution of an algorithmic problem and that teachers should demonstrate the complete analysis and design processes instead of just final products.

2.4.2.3 *Recursion*

Levy and Lapidot (2002) conducted a study to determine how novices see recursion as an interdisciplinary concept and identified patterns of students' expressions and ways of thinking when creating recursive descriptions using a natural language. The study findings indicated that class discourse plays an important role in helping students to understand recursion and in helping educators to understand students' conceptual schemes.

Haberman and Averbuch's (2002) study focused on students' conceptions of the base case as an integral component of a recursive algorithm. They found that students have difficulties identifying base cases. They handle redundant base cases, ignore boundary values and degenerated cases, avoid out-of-range values, and may not even define any base cases when formulating recursive algorithms. They suggested that teachers should discuss different aspects of the base case concept, such as declarative and procedural aspects of categorizing and handling base cases as part of formulating recursion. In addition, Haberman (2004b) found that students who learned recursion in logic programming using a declarative approach before learning it in a procedural paradigm, constructed more adequate mental models of recursion and were therefore better able to use recursion as a tool for knowledge representation.

2.4.2.4 *Abstraction and Abstract Data Types*

Many studies have investigated high-school students' perceptions of abstraction and abstract data types. Gal-Ezer and Zeldes (2000) for example, found that students who studied a software design unit, acquired technical skills that improved their handling of problems requiring the definition and use of abstract data types, even though many of them did not fully understand the concept itself. Haberman, Shapiro, and Scherz's (2002) study on the cognitive aspects of using abstract data types as tools for knowledge representation and problem solving in the declarative logic programming paradigm revealed that students' perceptions of the abstract data type concept varied as did their strategies for program development. Perhaps surprisingly, Haberman

(2004a) also found that high school students felt more comfortable than undergraduate students with algorithms with low-level abstraction.

2.4.2.5 Non-determinism

Armoni and Gal-Ezer's (2003) study of high school students' perceptions of the advanced abstract concept of non-determinism showed that students had a marked tendency to use the deterministic model independently of the given problem, indicating that they had only a partial understanding of the non-deterministic model. Based on the study results, the researchers suggested that the teaching process should emphasize both the theoretical and technical aspects of computer science.

2.4.2.6 Correctness

Haberman and Averbuch (2002) found significant evidence of students' misconceptions about correctness. In their study, students indicated that recursive algorithms were correct even though they did not handle every possible instance of the problem while at the same time they indicated that an algorithm was incorrect when it handled imperceptible, though relevant, base cases. This conclusion was further supported by Ben-David Kolikant (2005). She found that, in contrast to the professional dichotomous definition of correctness, students understood correctness as a relative property of the program and therefore tolerated errors. For example, students who produce incorrect programs might describe the programs as mostly correct; or they might consider programs as correct when they produce the expected output, yet in addition, they produce more unexpected output. Recent research by Haberman, Averbuch, and Ginat (2005) also determined that students hold the misconception that a short algorithm is not satisfactory if it neither represents the story described in the problem nor the problem's analysis process. Hence, students might not consider short concise algorithms as satisfactory solutions, even though they correctly yield the desired I/O relationship.

2.4.2.7 Algorithm Efficiency

Ginat's (1996) findings indicated that students' solutions to algorithmic problems varied considerably in terms of efficiency, reflecting different levels of insight into the problems. He found that fruitful class discussions of different solutions to a given problem regarding efficiency widened students' repertoire of possible solutions and made them realize the importance of analysis and planning in developing efficient solutions. Gal-Ezer and Zur (2002) also investigated high school students' perceptions of the concept of algorithm efficiency. They found that most beginners (grade 10 students) had much more difficulty internalizing the concept of algorithm efficiency than the 11th graders, because of related misconceptions such as "the more variables, the more execution time." Grade 11 students who opted for computer science, internalized the concept and demonstrated the ability to write efficient programs to solve simple algorithmic problems. When solving complex algorithmic problems, however, these students tended to write insufficient programs because of their inadequate ability to analyze the problem. Gal-Ezer and Zur therefore recommended that teachers should invest more effort teaching students mathematical analysis before writing programs.

2.4.2.8 Program Testing

Ben-David Kolikant (2005) investigated high school students' work habits and conceptions of testing. The study findings indicated that the students used inadequate methods of testing, and that their definitions of systematic testing were inherently different from those of professionals. Ben-David Kolikant and Pollack (2004a) found that the students' norm is to produce working, but not necessarily error-free, programs and to argue for their correctness

solely on the basis of a few executions. They suggested the integration of explanatory proofs as a means of establishing norms of precision in the students' practices and methods of communication. Haberman, Averbuch, and Ginat (2005) further supported this conclusion and argued that explicit rules of oral and written discourse should be elaborated in order to establish reliable and coherent student-teacher and student-peer communication.

2.4.3 Pedagogical Approaches

In addition to examinations of principles and concepts in high school computer science instruction, considerable research effort has been spent attempting to identify the most effective pedagogical approaches for addressing the principles and concepts that must be taught. Much of the research focuses on the use of specific programming languages and/or integrated development environments. In this section, however, we will concentrate on a selection of key studies that focus on instructional methodologies.

2.4.3.1 *Problem-Solving Strategies*

Ginat (2000) implemented a problem-solving approach that used colorful and attractive challenges and games involving physical objects at the primary stage of problem analysis to prompt students to look for the problem's characteristics "from various angles, in different ways, and for diverse tasks" (p. 81). They argued that this approach allowed high school students to develop "a set of values and perspectives that enhanced their mathematical and computer science point of view" (Ginat, 2000, p.84).

Muller (2005) investigated the influence of pattern-oriented-instruction on the development of problem-solving skills. The preliminary results of the study showed that students who had acquired a repertoire of patterns achieved greater insight into a problem's essence in a shorter time and were able to develop solution ideas more easily and with less distraction from the details of a problem's context. Moreover, pattern-oriented-instruction improved the written and verbal formulation of ideas of both teachers and students).

2.4.3.2 *Learning Different Programming Paradigms*

Haberman and Averbuch (2002) found negative transfer among students transitioning from the declarative logic programming paradigm to a procedural paradigm. Their findings indicated that students who learn list processing in Prolog tended to ignore boundary cases that control the termination of recursive computation when developing procedural algorithms. In contrast, Haberman (2004b) found that students who learned recursion in logic programming according to a declarative approach, before learning it in a procedural paradigm, constructed better mental models of recursion than students who were acquainted with recursion in procedural programming only.

Paz and Lapidot (2004) investigated the performance of high school students while they were first exposed to the functional programming paradigm. The most prominent finding in the study concerned the students' conception that a list processing function may change the value of its input parameter.

Ragonis and Ben-Ari's (2005) study focused on how novice programmers learn object-oriented programming. Their investigation into the concepts held by novices revealed many difficulties and erroneous conceptions that the researchers categorized into four primary categories:

- class versus object,
- instantiation and constructors,
- simple versus composed classes, and
- program flow.

The study findings further indicated that when the instructor provided a detailed description of the difficulties and conceptions together with an integrated and holistic view supported by specific

examples, authentic episodes, and interpretations, the students developed a much better understanding of the basic principles of object-oriented programming.

2.4.3.3 *Software Design and Project Development*

Sims-Knight and Upchurch's (1993) study found that when instructors taught software design before a specific programming language, students grasped the basic concepts necessary to create high-level designs even though they had no programming experience. Gal-Ezer and Zeldes (2000), however, found that high school students who studied software design exhibited difficulties in designing general top-down solutions for a given problem; and instead, they preferred to deal with specific examples. The students, however, were able to reuse general structures to distinguish complex tools from basic tools.

Collofello's (2002) study indicated that students who had access to a simulated environment in which they could learn and practice software development acquired their understanding of the software development process as a gradual process composed of many steps. Moreover, they were better at documenting the development requirements for their projects, employing use cases, and in using systematic testing and inspection techniques.

Pollack and Scherz (2005) investigated the influence of supportive learning materials on high school students' motivation, performance, and final products as they developed projects in computer science. The study findings indicated that students who tended to perceive projects as a school activity were mainly motivated by outside rewards such as the projects' external assessment. Students who did not use supporting materials for project-based learning and instruction used a bricolage approach to develop their projects instead. They also tended to modify the original problem according to their ability to develop a proper project. In contrast, students who used support materials for project-based learning showed a greater commitment to the project and submitted a product that referred to the original problem. Scherz and Haberman (2005) also found students who used problem-solving organizing tools in developing their projects were more likely to use abstract data types that resulted in a structured and well-organized development process.

2.5 WHOM SHOULD WE TEACH?

The current structure of high school education provides many indicators as to the value placed upon a foundational understanding of the traditional sciences as part of the required knowledge of every educated citizen. Physics, biology, and chemistry have long been required elements of high school student learning. For computer science, however, the discussion of its increasing importance as part of the required knowledge base of every citizen in our increasingly technological world, has been sadly absent, and hence is long overdue. Lee (2004), for example, argued that academic computer science must "re-educate the public-at-large (especially children) on the scientific foundations and fundamental questions of the field." Guzdial and Solloway (2003) also suggested, "Students should emerge from an introductory course with a sense about what's interesting in the field, and they should have some practical knowledge that they can apply in their fields" (p. 5). Unfortunately, however, research has demonstrated computer science's continuing inability to attract young women and minority students to the discipline in representative numbers and to keep them there beyond the introductory level.

Even in countries where there is a commitment to providing computer science education for all high school students, research has identified two long-standing barriers to equity. Specifically, young women and minority students continue to be disproportionately under-represented in computer science education. This pattern is most clearly exemplified in the United States by the demographic breakdown of students taking Advanced Placement (AP) computer science. The College Board (2005)

reported that in 2004 only 15% of the students taking the AP computer science exams were female. In addition, fewer than 5% of students of either sex self-identified as Latino/Latina and fewer than 3% were African American.

To date there has been a considerable amount of educational research dedicated to determining why this under-representation persists in computer science and not in other sciences. Fisher and Margolis (2002), for example, found that the context of computing is often very important for women and that more women than men link their interest in computer science to other arenas. They therefore recommended that both the curriculum and the culture are changed to ensure that young women do not perceive that they will succeed only if they "model themselves after the stereotypical male computer science student" (p. 80). As already mentioned, research by Moorman and Johnson (2003) provided further support for Fisher and Margolis' conclusions. They found that "Women in computing still perceive themselves as strangers in a strange, male-dominated land" (p. 193). Moorman and Johnson (2003) also found that both male and female students continue to see computer science as a primarily "male" field and to make their career choices accordingly. They suggested that single-sex summer workshops for high school girls and mother-daughter computer science clubs for elementary school girls would help to combat this perception.

Moorman and Johnson's call for single-sex educational opportunities was consistent with an earlier study by Crombie, Abarbanel and Anderson (2000). Their study on girl-only learning settings found that female enrollment significantly increased in all-female computer science classes. The girls in their study also exhibited a higher level of confidence in their own skills than did girls in mixed gender classes. These results were also supported by Graham and Latulipe (2003). As a result of their investigation at the University of Waterloo, they developed an all-female computer science seminar for high school girls to encourage female students to enter computer science. This program focused on showing young women a real-world perspective on computer science and providing them with positive role models to dispel the negative stereotypes.

Research has also provided significant evidence that minority students have been similarly disaffected from computer science, both as a discipline and as a career area. Although policymakers and industry leaders encourage minority participation in science and technology academic studies with the intention of increasing the number of minority students who enter, are retained in, and succeed in high-tech careers, the research indicates that a very low percentage of minority students intend to major in this field. Payton (2003), for example, found that African-American teenagers tended to avoid majoring in computer science, information science, or information technology.

2.6 EXEMPLARY TEACHERS

While it is clear that student attitudes are an important factor in students choosing to study computer science in high school, it is also clear that teachers are the corner stone of successful curriculum implementation. As a result, there has been considerable research relating to the development of appropriate computer science teacher training programs (Gal-Ezer et al., 1995; Lapidot & Hazzan, 2003; Tucker et al., 2004; Dorninger, 2005). The general consensus of this research, as Gal-Ezer and Harel (1998) note, is that high school computer science teachers should be trained to convey technical knowledge correctly and reliably, to teach skills, to provide perspective, and to infuse the students with interest, curiosity, and enthusiasm. Recruiting and retraining competent high school computer science teachers, however, has proven extremely problematic due to a lack of adequate teacher training programs and the relatively low pay compared to that of industrial computer scientists (Poirot, 1979; Poirot, Taylor, & Norris, 1988).

Researchers such as Poirot (1979) have therefore identified several levels of educational intervention

and support needed to ensure an adequate supply of highly trained computer science teachers. These include:

- conducting training institutes for in-service computer science secondary teachers,
- implementing a service course within the computer science curriculum designed especially for prospective computer science secondary teacher training, and
- implementing teacher certification programs in computer science.

The following sections examine the research relating to both pre-service and in-service teacher training in more detail.

2.6.1 Pre-service Teacher Training

The existing research highlighting the link between student success at the college/university level and the methodology used to teach high school students (Taylor & Mounfield, 1991) has raised fundamental questions concerning pre-service computer science teacher education. According to Lapidot and Hazzan (2003) the pressing questions articulated by this research include the following:

- Which appropriate frameworks would fit a *Methods for Teaching Computer Science in High School (MTCS)* course?
- Which topics should be included in such courses?
- Which kinds of activities would be best suited for the preparation of future computer science teachers?

In an effort to address similar questions, Gal-Ezer and Harel (1998) recommended that computer science teachers should acquire both technical and pedagogical knowledge: "Beyond the mastery of core CS material, good CS educators should also be familiar with a significant body of material that will expand their perspectives of the field, and consequently, enhance the quality of their teaching" (p. 77). This conclusion was supported by Lapidot and Hazzan (2003) who noted that "a merger of computer science with pedagogy into one amalgam of content and pedagogy" (p. 30) would prepare teachers to better employ general pedagogical principles as well as teaching methods in the context of computer science education.

2.6.1.1 *Technical Knowledge*

According to Gal-Ezer and Harel (1998) and Poirot (1979), high school computer science teachers should have a thorough formal background in computer science (on an academic level). Teachers must also have a broad overview of the pertinent areas of computer science, and to have acquired a bird's-eye view of the discipline: "For those desiring a more thorough coverage of a particular topic, existing computer science coursework would be required. The course content of a computer education course must not only include computer related material, but also the motivation for covering each topic" (Poirot, 1979, p. 102).

2.6.1.2 *Pedagogical Knowledge*

Many computer science educators lack formal training in education and are unfamiliar with educational theories; hence, they rarely attempt to apply these theories in the computer science classroom (Gal-Ezer & Harel, 1998; Ben-Ari, 2004). To be best qualified, computer science high school teachers should be certified and offered courses in computer science education in addition to regular computer science courses (Schollmeyer, 1996). Gal-Ezer and Harel (1998) additionally recommend that several topics that would expand teachers' perspectives of the discipline should be part of a computer science teacher's education. These would include:

- a history of computer science,
- the nature of computer science and its relationship to other disciplines,

- various computer science curricula,
- pedagogical knowledge related to teaching computer science, and
- the use of tools and aids in teaching the subject.

They also suggest integrating these subjects into specially tailored in-service training programs for high school teachers, especially those who lack a proper pre-service computer science and pedagogical education. Based on their experience of teaching the MTCS course to prospective teachers, Lapidot and Hazzan (2003) additionally recommended optional course frameworks and corresponding implementations arranged as clusters of optional activities. For example, they suggested an active learning-based teaching model that would support the construction of computer science teachers' professional perception (Hazzan & Lapidot, 2004a).

Tucker et al. (2004) argued that as part of their pre-service training, prospective computer science teachers must also practice the teaching of computer science in schools before becoming actual computer science teachers. This argument was further supported by Hazzan and Lapidot (2004b), who suggested that teaching practice should be organized in a way that supports bridging gaps between theoretical knowledge and actual performance:

> no matter how much this topic is discussed in the MTCS course, the actual implementation of different teaching methods in real teaching situations, together with a reflection process that follows, can improve the prospective teachers' understanding with respect to this topic. (p. 48)

They also noted that the illustration of situations that take place during the practice may help bridge the gap between the theory (the MTCS course), reality (what actually happens in schools), and the university mentor's perspective (academia).

2.6.1.3 *Socio-cultural Knowledge*

There is also a small but interesting body of research indicating that prospective computer science teachers should be educated to become part of a collaborative computer science teachers' community of practice (Haberman, Lev, & Langley, 2003; Ben-David Kolikant & Pollack, 2004b). Specifically, they should be motivated to learn from the experience of experts and experienced teachers and to seek their counseling. Moreover, they should be guided to contact supportive groups such as local and national teacher associations and to attend their activities (Lapidot, 2002). In this way, it is suggested that teachers will gradually proceed towards full-fledged membership in the computer science teachers' community of practice.

2.6.2 On-going Professional Development for Teachers

The rapid change of programming languages and tools for teaching computer science makes it exceedingly difficult for teachers to remain current in this discipline. This challenge makes computer science less attractive to qualified teachers and often leads to defections to other subject areas (Roberts, 2004). Thus, a tremendous effort is needed to support teachers in the adaptation to new programming languages and tools for teaching computer science, the adoption and implementation of new curricula, and in the constant enhancement of their pedagogical approaches.

Assimilating a new computer science curriculum requires appropriate in-service training aimed at introducing the curriculum and its didactic approach (Haberman & Ginat, 1999). In addition, teachers require on-going support in all aspects of classroom implementation of the curriculum. According to Haberman, Lev, and Langley (2003) the assimilation of a new curriculum "should be accompanied by the establishment of a collaborative 'learning and creating' community of teachers. This community will begin its activity within an organized guided course and continue with intra-school activity and inter-school cooperation" (p. 144).

2.6.3 The Role of Professional Associations in Supporting Teachers

In addition to occasional projects of massive widespread training, national and regional teacher associations are playing a considerable and growing role in both the development and support of new curricula. They are also, especially in the case of computer science teachers, becoming the primary providers of professional development opportunities (Driscoll, 1987; Inos & Quigley, 1995; Adagian, 1996). As Harris (1987) noted, participation in professional associations has a direct impact on professional effectiveness because it provides many direct benefits to teachers. It:

- provides new expertise through professional development workshops,
- establishes networking relationships that provide opportunities to share curricula,
- develops stronger personal commitment to students and the academic discipline, and
- increases the realization of the importance of connections with business and industry.

Professional associations also provide exposure to new ideas, either through organized events such as conferences and workshops or through professional publications (Bell, 1983). As Romberg and Middleton (1995) argued, "without programs that foster collaboration, teachers would not be able to discover new teaching methods and ways of dealing with students and administration that their colleagues can and do provide" (p. 175).

Professional educational associations also provide essential support for specific academic disciplines and play a key role in the development of leadership within those disciplines. According to Jeffrey (1996) professional teacher associations support individual academic disciplines by:

- encouraging the testing of new initiatives,
- sharing expertise,
- satisfying an almost insatiable need for discipline-based professional development,
- identifying key research needs,
- communicating the perspective of those in a particular discipline to the wider educational community,
- recognizing trends in discipline content, and
- putting strategies in place to ensure that equitable standards are maintained.

The Computer Science Teachers Association (CSTA) for example, is a membership organization that supports and promotes the teaching of computer science and other computing disciplines at the K–12 level. Established in 2004, CSTA has already built a strong system of partnerships and carried out a number of successful projects (development of the ACM *Model Curriculum for K–12 Computer Science Education,* the Java Engagement for Teacher Training project, five Computer Science and Information Technology symposia) that have given it credibility and forged strong ties with the community of computer science educators.

After examining the current research and experiential classroom and administrative practices, the CSTA Board of Directors identified the following key components required to facilitate much needed improvements to K–12 computer science education:

1. **Communication/Partnership:** Effective change requires the engagement and support of teachers, principals, school board staff, guidance counselors, administrators, parents, college faculty, College of Education faculty, legislators, and key industry partners.
2. **Curriculum Standards:** Consultation on, publication of, endorsement of, and adoption of curriculum guidelines are needed to establish the required body of knowledge and improve consistency in classrooms nationally.
3. **Curriculum Support:** Models of success and the development and dissemination of teaching and learning materials help ensure the adoption of curriculum standards and address critical resource shortages. These must be supported by accessible and relevant professional development for teachers.
4. **Research:** Policy-makers (administrators and legislators) require data to convince them that

it is time for a change and that proposed change mechanisms will work.

5. **Teacher Certification Standards:** Ensuring that there are sufficient numbers of qualified teachers and that teacher certification standards are consistent from state-to-state will improve pedagogy and course content nationally.

Like many other such associations internationally, CSTA has taken on the role of not only advocating for the place of computer science within the larger high school curriculum, but also providing the resources and support that practitioners require to continue to improve their teaching (http://csta.acm.org/).

Machshava, (the Israeli National Center for High School Computer Science Teachers) (http://cse.proj.ac.il), has assumed a similar role for Israeli teachers (Lapidot, 2002). Its initiatives are directed at similar goals, including:

- fostering professional leadership of computer science teachers;
- helping create a professional community of computer science teachers; identifying computer science teachers' needs;
- supporting local teacher centers; and
- collecting and distributing computer science education knowledge and experience.

2.7 CONCLUSION

For many years, there has been considerable debate over the place of computer science education in the high school curriculum (Stephenson, 2002). While researchers and educators may hold differing opinions concerning when a program of computer science education should be implemented and how it should be taught, the research consistently supports the conclusion that, like any other academic discipline, it must be engaging and rigorous, or as Gal-Ezer et al. (1995) concluded, high school computer science "must be challenging, in the sense that it will not only teach the foundations, but will also relate them to the practical side of computing, and it should train the students to deal with intellectually demanding tasks" (Gal-Ezer et al., 1995). Deciding what and how to teach is, however, just one element in ensuring that even the best curriculum is actually implemented and supported over time. The next chapter will, therefore, examine the multi-level issues that affect successful curriculum implementation.

2.8 REFERENCES

Adajian, L. B. (1996). Professional communities: Teachers supporting teachers. *Mathematics Teacher, 89*(4), 321–364.

Almstrum, V. L., Hazzan, O., & Ginat D. (Eds.). (2004). Special issue on import/export relationships to computer science education research. *Computer Science Education, 14*(4).

Armoni, M., & Gal-Ezer, J. (2003). Non-determinism in CS high school curricula. *Proceedings of the FIE 2003 Conference,* Boulder, CO, F2C–18 – F2C–23.

Bell, D. (1983). Promoting business education through teacher participation in professional associations. *Business Education Forum/Yearbook, 37*(8), 48–51.

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching, 20*(1), 45–73.

Ben-Ari, M. (2002). From theory to experiment to practice in CS education. Paper presented at *Kolin Kolistelut—Koli Calling: 2nd Annual Finnish/Baltic Sea Conference on Computer Science Education,* Koli, Finland, October 2002.

Ben-Ari, M. (2004). Situated learning in computer science education, *Computer Science Education, 14*(2), 85–100.

Ben-David Kolikant, Y. (2004). Learning concurrency: Evolution of students' understanding of

synchronization. *The International Journal of Human Computers Studies, 60*(2), 243–268.

Ben-David Kolikant, Y., & Pollack, S. (2004a). Establishing computer science professional norms among high school students. *Computer Science Education, 14*(1), 21–35.

Ben-David Kolikant, Y., & Pollack, S. (2004b). Community-oriented pedagogy for inservice CS teacher training. *Proceedings of the ITiCSE 2004 Conference,* Leeds, United Kingdom, 191–195.

Ben-David Kolikant, Y. (2005). Students' alternative standards for correctness. *Proceedings of the First International Computing Education Research Workshop,* October 1–2, 2005, University of Washington, Seattle, WA, 37–43.

Bergin, S., & Reilly, R. (2005). Programming: Factors that influence success. *Proceedings of the SIGCSE 2005 Conference,* St. Louis, Missouri, USA, 411–415.

Böszörmenyi, L. (2005). Teaching: People to people—about people: A plea for the historic and human view. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 93–103.

Campbell, P. F., & McCabe, G. P. (1984). Predicting the success of freshmen in a computer science major. *Communications of the ACM, 27*(11), 1108–1113.

Cantwell Wilson, B., & Shrock, S. (2001). Contributing to success in an introductory computer science course: A study of twelve factors. *SIGCSE Bulletin, 33*(1), 184–188.

Cantwell Wilson, B. (2002). A study of factors promoting success in computer science including gender differences. *Computer Science Education, 12*(1–2), 141–164.

Cartelli, A. C. (2002). Computer science education in Italy: A survey. *SIGCSE Bulletin, 34*(4), 36–39.

Computer Science Teachers Association. (2005). Results of the national secondary computer science survey. Retrieved October 16, 2005, from http://csta.acm.org/Research/sub/CSTANationalSurvey2004.html

Collofello, J. S. (2002). Creation, development and evaluation of an innovative secondary school software development curriculum module. *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference,* Boston, MA.

Crombie, G., Abarbanel, T., & Anderson, C. (2000). All-female computer science: The positive effects of single gender classes. *The Science Teacher,* March 2000, 40–43.

Dagienë, V. (2005). Teaching information technology in general education: Challenges and perspectives. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 53–64.

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1988). *Draft report of the ACM Task Force on the Core of Computer Science.* New York, NY: Association for Computing Machinery.

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a discipline. *Communications of the ACM, 32*(1), 9–23.

Denning, P. J. (2004). Great principles in computing curricula. *Proceedings of the SIGCSE 2004 Conference,* Norfolk, Virginia, USA, 336–341.

Dorninger, C. (2005). Educational standards in school informatics in Austria. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 66–69.

du Boulay, B., O'Shea, T., & Monk, J. (1989). The black box inside the glass box: Presenting computing concepts to novices. In E. Soloway & J. C. Spohrer (Eds.). *Studying the novice programmer* (pp. 431–446). Hillsdale, NJ: Lawrence Erlbaum Associates.

Driscoll, M. (1987). *Ten case studies from a study of exemplary mathematics programs.* Reston, VA: National Council of Teachers of Mathematics.

Eurydice (Information Network on Education in Europe). (2005). *Information and communication technologies in education.* Retrieved January 25, 2005 from http://www.eurydice.org/Documents/Bibliographie/en/frameset_biblio_refs_en.html

Fadi, P. D., & McHugh, J. A. (2000). Problem-solving methodologies and the development of critical thinking skills. *Journal of Computer Science Education, 14*(1&2), 6–12.

Fadi, P. D., & McHugh, J. A. (2001). Prototype software development tools for beginning programming. *Journal of Computer Science Education, 14*(3&4), 14–20.

Felleisen, M., Findler, R. B., Flatt, M., & Krishnamurthi, S. (2002). The structure and interpretation of the computer science curriculum. *Journal of Functional Programming, 14*(4), 365–378.

Fincher, S. (1999). What are we doing when we teach programming? *Proceedings of the FIE1999 Conference,* San Juan, Puerto Rico, 12a4–1–12a4–4.

Fincher, S., & Petre, M. (2004). *Computer science education research.* London, UK: Routledge Falmer.

Fisher, A., & Margolis, J. (2002). Unlocking the clubhouse: The Carnegie Mellon experience. *SIGCSE Bulletin, 34*(2), 79–83.

Foley, J. (2004). Old challenges, new opportunities. *Computing Research News, 16*(4). Retrieved December 1, 2005 from http://www.cra.org/CRN/articles/sept04/foley.html

Franklin, R. (1987). What academic impact are high school computing courses having on the entry-level computer science curriculum? *Proceedings of the SIGCSE 1987 Conference,* St. Louis, Missouri, USA, 253–256.

Gal-Ezer, J., Beeri, C., Harel, D., & Yehudai, A. (1995). A high school program in computer science. *Computer, 28*(10), 73–80.

Gal-Ezer, J., & Harel, D. (1998). What (else) should CS educators know? *Communications of the ACM, 41*(9), 77–84.

Gal-Ezer, J., & Zeldes, A. (2000). Teaching software designing skills. *Computer Science Education, 10*(1), 25–38.

Gal-Ezer, J., & Zur, E. (2002). The concept of "algorithm efficiency" in the high school CS curriculum. *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference,* Boston, MA, T2C1–T2C6.

Gal-Ezer, J., Vilner, T., & Zur, E. (2003). *Characteristics of students who failed (or succeeded) the introductory CS course.* Paper presented at the *FIEE 2003 Conference.* Boulder, CO. Retrieved December 4, 2005 from http://fie.engrng.pitt.edu/fie2003

Ginat, D. (1996). Efficiency of algorithms for programming beginners. *Proceedings of the SIGCSE 1996 Conference.* Philadelphia, PA, 256–260.

Ginat, D. (2000). Colorful examples for elaborating exploration of regularities in high school CS1. *Proceedings of ITiCSE 2000,* Helsinki, Finland, 81–84.

Goldweber, M., Fincher, S., Clark, M., & Pears. A. (2004). The relationships between CS education research and the SIGCSE community. *SIGCSE Bulletin, 36*(1), 147–148.

Graham, S., & Latulipe, C. (2003). CS girls rock: Sparking interest in computer science and debunking the stereotypes, *Proceedings of the SIGCSE 2003 Conference,* Reno, Nevada, USA, 322–326.

Grandell, L. (2005). High school students learning university level computer science on the web—a case study of the DASK-model. *Journal of Information Technology Education, 4*, 207–218.

Greening, T. (1998). Computer Science: Through the eyes of potential students, *Proceedings of the ACSE 1998 Conference,* Brisbane, Australia, 145–154.

Gries, D. (2002). Where is programming methodology these days? *SIGCSE Bulletin, 34*(4), 5–7.

Gurbiel, E., Hardt-Olejniczak, G., & Kolczyk, E. (2005). Informatics and ICT in the Polish education system. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 93–103.

Gupta, U. G., & Houtz, L. E. (2000). High school students' perceptions of information technology skills and careers. *Journal of Industrial Technology, 16*(4), 2–8.

Guzdial, M. and Soloway, E. (2003). Computer science is more important than calculus: The challenge of living up to our potential. *SIGCSE Bulletin, 35*(2), 5–8.

Haberman, B., & Ginat, D. (1999). Distance learning model with local workshop sessions applied to in-service teacher training, *Proceedings of the ITiCSE 1999 Conference,* Krakow, Poland, 64–67.

Haberman, B., & Ben-David Kolikant, Y. (2001). Activating 'black boxes' instead of opening 'zippers'—A method of teaching novices basic CS concepts. *Proceedings of the ITiCSE 2001 Conference,* Dublin, Ireland, 41–44.

Haberman, B., & Averbuch, H. (2002). The case of base cases: Why are they so difficult to recognize? Student difficulties with recursion. *Proceedings of the ITiCSE 2002 Conference,* Aarhus, Denmark, 84–88.

Haberman, B., Shapiro, E., & Scherz, Z. (2002). Are black boxes transparent?—High school students' strategies of using abstract data types. *Journal of Educational Computing Research, 27*(4), 411–436.

Haberman, B., Lev, L., & Langley, D. (2003). Action research as a tool for promoting teacher awareness of students' conceptual understanding. *Proceedings of the ITiCSE 2003 Conference,* Thessaloniki, Greece, 144–148.

Haberman, B. (2004a). High school students' attitudes regarding procedural abstraction: Special issue devoted to recent research projects of secondary informatics education. *Education and Information Technologies, 9*(2), 131–145.

Haberman, B. (2004b). How learning logic programming affects recursion comprehension. *Computer Science Education, 14*(1), 37–53.

Haberman, B., Averbuch, H., &. Ginat, D. (2005). Is it really an algorithm?—The need for explicit discourse. *Proceedings of the ITiCSE 2005 Conference,* Lisbon, Portugal, 74–78.

Hagan, D., & Markham, S. (2000). Does it help to have some programming experience before beginning a computing degree program? *SIGCSE Bulletin, 32*(3), 25–28.

Harris, J. W. (1987). Teacher and leader—why we must be both. *Vocational Education Journal, 62*(5), 26–28.

Hazzan O., Impagliazzo, J., Lister, R., & Schocken, S. (2005). Using history of computing to address problems and opportunities, *Proceedings of the SIGCSE 2005 Conference,* St. Louis, MO, 126–127.

Hazzan, O., & Lapidot, T. (2004a). Construction of a professional perception in the "Methods of Teaching Computer Science" course. *SIGCSE Bulletin, 36*(2), 57–61.

Hazzan, O., & Lapidot, T. (2004b). The practicum in computer science education: Bridging the gap between theoretical knowledge and actual performance. *SIGCSE Bulletin, 36*(4), 47–51.

Holmboe, C., McIver, L., & George, C. (2001). Research agenda for computer science education. *Proceedings of the 13th Workshop of the Psychology of Programming Interest Group,* Bournemouth, UK, 207–233.

Inos, R. H., & Quigley, M. A. (1995). *Synthesis of the research on educational change. Part 4: The teacher's role.* Honolulu, HI: Pacific Region Educational Laboratory.

International Society for Technology in Education (ISTE). (2002) *National educational technology standards for students.* Retrieved December 1, 2005, from http://www.iste.org/netsImpagliazzo, J., & Lee, J. A. N. (2004). Using computing history to enhance teaching. *Proceedings of the History of Computing in Education 2004 Conference,* Toulouse, France, 165–176.

Jeffrey, N. (1996). The responsibility of professional associations. Paper presented at *A Meeting of the Minds: ITEC Virtual Conference 1996,* Australia.

Kavander, T., & Salakoski, T. (2004). Where have all the flowers gone? Computer science education in general upper secondary school. *Proceedings of the Kolin Kolistelut—Koli Calling Conference,* Koli, Finland, 112–115.

Kuznetsov, A. A., & Beshenkov, S. A. (2005). Russian educational standards of informatics and informatics technologies (ICT): Aims, content, perspectives. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 70–73.

Lapidot, T., & Hazzan, O. (2003). Methods of teaching computer science courses for prospective teachers. *SIGCSE Bulletin, 35*(4), 29–34.

Latulipe, C., & Graham, S. (2005). Degaming the curriculum: A failed experiment. Retrieved December 10, 2005, from: http://hci.uwaterloo.ca/students/clatulip/research.html

Lee, P. (2004). *The computer science brain drain: A call to revitalize computer science education.* Retrieved April 23, 2004, from https://www.wiki.cs.cmu.edu/public/uploads/Main/it-talent.pdf

Levy, D., & Lapidot, T. (2002). Shared terminology, private syntax: The case of recursive descriptions. *Proceedings of the ITiCSE 2002 Conference,* Aarhus, Denmark, 89–93.

Loidl, S., Muhlbacher, J., & Scauer, H. (2005). Preparatory knowledge: Propaedeutic in informatics. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 93–103.

Merritt, S. (1995). Reflections of a computer scientist

for teachers and teacher educators. In J. D. Tinsley and T. J. van Weert (Eds.). *Proceedings of the World Conference on Computers in Education* VI (4790486). London: Chapman and Hall for the International Federation of Information Processing.

Micheuz, P. (2005). 20 years of computers and informatics in Austria's secondary academic schools. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives,* ISSEP 2005, Klagenfurt, Austria, March/April 2005, 20–31.

Mitchell, W. (2002). Information technology education: One state's experience. *Journal of Computing in Small Colleges, 17*(4), 123–132.

Mittermeir, R. T. (Ed.) (2005). *From computer literacy to informatics fundamentals. International Conference on Informatics in Secondary Schools-—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005.

Moorman, P., & Johnson, E. (2003). Still a stranger here: Attitudes among secondary school students towards computer science. *Proceedings of the ITiCSE 2003 Conference,* Thessaloniki, Greece, 193–197.

Morris, J. H. and Lee, P. (2004). The incredibly shrinking pipeline is not just for women anymore. *Computing Research News, 16*(1), 20.

Muller, O., Haberman, B., & Averbuch, H. (2004). (An almost) pedagogical pattern for pattern-based problem-solving instruction. *Proceedings of the ITiCSE 2004 Conference,* Leeds, United Kingdom, 102–106.

Muller, O. (2005). Pattern oriented instruction and the enhancement of analogical reasoning. *Proceedings of The First International Computing Education Research Workshop*, October 1–2, 2005. Seattle, WA, 57–67.

National Research Council Committee on Information Technology Literacy. (1999). *Being fluent with information technology.* Washington, DC: National Academy Press.

O'Lander, R. (1996). Factors effecting high school student's choice of computer science as a major. *Proceedings of the Symposium on Computers and the Quality of Life,* Philadelphia, Pennsylvania, USA, 25–31.

Paz, T., & Lapidot, T. (2004). Emergence of automated assignment conceptions in a functional programming course. *Proceedings of the ITiCSE 2004 Conference,* Leeds, United Kingdom, 181–185.

Payton, F. C. (2003). Rethinking the digital divide. *Communications of the ACM, 46*(6), 89–91.

Peckham, J., DiPippo, L., Reynolds, J., Paris, J., Monte, P., & Constantinidis, P.A. (2000), First course in computer science: The discipline is more than programming. *Journal of Computing in Small Colleges, 15*(5), 223–230.

Perkins, D., Schwartz, S., & Simmons, R. (1988). Instructional strategies for the problems of novice programmers. In R. E. Mayer (Ed.), *Teaching and learning computer programming* (pp. 153–178). Hillsdale, NJ: Lawrence Erlbaum Associates.

Pham, B. (1997). The changing curriculum of computing and information technology in Australia. *Proceedings of the ACSE 1997 Conference,* Melbourne, Australia, 149–154.

Poirot, J. L. (1979). Computer education in the secondary school: Problems and solutions. *Proceedings of the SIGCSE 1979 Conference,* Dayton, Ohio, 101–104.

Poirot, J. L., Taylor, H. G., & Norris, C. A. (1988). Retraining teachers to teach high school computer science. *Communications of the ACM, 37*(7), 912–917.

Pollack, S., & Scherz, Z. (2005). Supporting project development in CS – the effect on intrinsic and extrinsic motivation. *Proceedings of the 10th PEG Conference,* Tampere, Finland, 143–148.

Ramberg, P. (1986). A new look at an old problem: Keys to success for computer science students. *ACM SIGCSE Bulletin, 18*(3), 36–39.

Ragonis, N., & Ben-Ari, M. (2005). A long-term investigation of the comprehension of OOP concepts by novices. *Computer Science Education,* 15(3), 203–221.

Rich, L., Perry, H., & Guzdial, M. (2004). A CS1 course designed to address interests of women. *Proceedings of the SIGCSE 2004 Conference,* Norfolk, Virginia, USA, 190–194.

Reiter, A. (2005). Incorporation of informatics in Austrian education: The project "Computer-Education-Society" in the school year 1984/85. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 4–19.

Roberts, E. (2004). The dream of a common language: The search for simplicity and stability in computer science education. *Proceedings of the SIGCSE 2004 Conference,* Norfolk, Virginia, USA, 115–119.

Roberts, E,. & Halopoff, G. (2005). Computer Science Teachers Association analysis of high school survey data. Retrieved December 2, 2005 from http://csta.acm.org/Research/sub/CSTANationalSurvey2004.html

Romberg, T. A., & Middleton, J. A. (1995). Conceptions of mathematics and mathematics education held by teachers. In N. Webb & T. A. Romberg (Eds.). *Collaboration as a process for reform.* New York: Teachers College Press.

Rountree, N., Vilner, T., Wilson, B., & Boyle, R. (2004). Predictors for success in studying CS, panel session. *Proceedings of the SIGCSE 2004 Conference,* Norfolk, Virginia, USA, 145–146.

Rosenthal, T., Suppes, P., & Ben-Zvi, N. (2002). Automated evaluation methods with attention to individual differences—A study of a computer-based course in C, *Proceedings of the ASEE/IEEE Frontiers in Education Conference,* Boston, MA.

Sabin, M., Higgs, B., Riabov, V., & Moreira, A. (2005). Designing a pre-college computing course. *Journal of Computing in Small Colleges, 20*(5), 123–132.

Scherz, Z., & Haberman, B. (2005). Mini-projects development in computer science—Students' use of organization tools. *Informatics in Education, 4,* 307–319.

Schollmeyer, M. (1996). Computer programming in high school vs. college. *Proceedings of the SIGCSE 1996 Conference,* Philadelphia, PA, USA, 378–382.

Sims-Knight, J. E., & Upchurch, R. L. (1993). Teaching software design: A new approach to high school computer science. Paper presented at *The Annual Meeting of the American Educational Research Association,* Atlanta, GA. Retrieved November 1, 2005 from http://www2.umassd.edu/cisw3/people/faculty/rupchurch/

Sleeman, D., Putnam, R. T., Baxter, J. A, & Kuspa, L. (1989). A summary of misconceptions of high school basic programmers. In E. Soloway & J. C. Spohrer (Eds.). S*tudying the novice programmer* (pp. 301–314). Hillsdale, NJ: Lawrence Erlbaum Associates.

Stephenson, C. (2002). High school computer science education. *Journal of Computer Science Education,* Annual, 30–40.

Stevenson, D. E. (1993). Science, computational science and computer science: At a crossroads. *Proceedings of the 1993 ACM Conference on Computer Science,* Indianapolis, IN, 7–14.

Stiller, E., & LeBlanc, C. (2003). Creating new computer science curricula for the new millennium. *Journal of Computing in Small Colleges, 18*(5), 198–209.

Taylor, H. G., & Mounfield, L. C. (1989). The effect of high school computer science, gender, and work on success in college computer science. *ACM SIGCSE Bulletin, 21*(1), 195–198.

Taylor, H. G., & Mounfield, L. C. (1991). An analysis of success factors in college computer science: High school methodology is a key element. *Journal of Research on Computing in Education, 24*(2), 240–245.

The College Board. (2005). *Advanced Placement report to the nation.* Retrieved December 22, 2005, from http://www.collegeboard.com/about/news_info/ap/2005/index.html

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C. and Verno, A. (2003). *A model curriculum for K–12 computer science: Report of the ACM K–12 Education Task Force Computer Science Curriculum Committee.* New York, NY: Association for Computing Machinery.

Tucker, A., McCowan, D., Deek, F., Stephenson, C., Jones, J., & Verno, A. (2004). Implementation challenges for K–12 computer science curriculum. *Proceedings of the SIGCSE 200404 Conference,* Norfolk, VA, 334–335.

Vandenberg, S., & Wollowski, M. (2000). Introducing computer science using a breadth-first approach and functional programming. *Proceedings of the SIGCSE 2000 Conference,* Austin, TX, USA, 180–184.

Ventura, P., & Ramamurthy, B. (2004). Wanted: CS1 students. No experience required. *Proceedings of the SIGCSE 2004 Conference,* Norfolk, Virginia, USA, 240–244.

CHAPTER THREE

USING FRANK'S FRAMEWORK TO EXPLORE THE DEVELOPMENT AND IMPLEMENTATION OF HIGH SCHOOL COMPUTER SCIENCE CURRICULA IN FIVE COUNTRIES

3.0 INTRODUCTION

In recent decades, computers have come to occupy a pivotal place in our work, personal, and home lives. Accordingly, the field of computer science has expanded to encompass the technical, innovative, conceptual, and psychological ramifications of the presence of the computer at work and at home. Much research (National Research Council, 1999) confirms the urgent need to improve the level of public understanding of computer science as an academic and professional field. In fact, the lack of a current commitment to computer science education at all levels is recognized as a major factor in the shortage of computer science professionals that continues to affect industries world-wide (Bureau of Labor Statistics, 2005). Many students receive their first exposure to many scientific disciplines in high school (secondary school). Lack of support and implementation of appropriate learning opportunities in computer science for students at this level therefore significantly affects the long-term national economic viability, not just in the high tech industries, but in all industries.

Although a significant amount of research has been conducted on teaching specific computer science concepts or using specific software tools for instruction, minimal attention has been paid to developing a solid set of instructional and content standards for pre-college computer science. As Tucker, Deek, Jones, McCowan, Stephenson, and Verno. (2003) noted:

> 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. (p. 3)

For this reason, the United States may find itself at considerable disadvantage when compared to other countries that have already implemented or are currently developing a comprehensive computer science curriculum for K–12 education.

This paper uses Frank's (1972) framework for critiquing educational policy and reform to analyze the transcript of a National Science Foundation-sponsored panel entitled *Finding a Computing Curriculum that Fits: the United States and Beyond* (Verno, Chiles, Gal-Ezer, Martin & Stephenson, 2005). The panel, which took place at the National Educational Computing Conference (NECC) in Philadelphia in June 2005, brought together curriculum experts from five countries: Canada, Israel, Scotland, South Africa, and the United States. Its goal was to discover elements that could be identified as contributing to the successful development and implementation of a national high school computer science curriculum.

3.1 METHODOLOGY

3.1.1 Method of Analysis

This paper collects the observations of five educators from five different countries who participated in a panel at NECC in 2005. Panel organizers selected the countries based upon their varying levels of

commitment to a national high school computer science curriculum. The individual panelists were selected on the strength of their individual and national curriculum development efforts, their personal activism in supporting excellence in computer science education nationally, and their ability to illuminate key implementation issues and strategies. They also represented various stakeholder constituencies, including researchers, curriculum developers, and educators. The following table contains a list of the panelists and the countries they represented.

PANELIST	COUNTRY/PROVINCE
Anita Verno (Chair)	United States
Chris Stephenson	Canada (Ontario)
Judith Gal-Ezer	Israel
Jackie Martin	Scotland
Michael Chiles	South Africa

Data for this study is drawn from a single source: a recorded and then professionally prepared transcript of the one-hour panel session. Employing the constant comparative method (Strauss, 1987), data analysis began with the first presentation and continued throughout the document. A coding framework for three main categories was drawn directly from the structure of the panel presentation (curriculum models, curriculum dissemination and implementation, and assessment). This framework was further revised and expanded as additional themes emerged. To ensure accuracy of reporting, the panelists and the panel chair reviewed the transcript drafts. The transcript was then coded, and 58 illustrative segments ranging in length from 30 to 187 words were copied from the word-processed transcript into a database file.

Analysis began with the grouping of similarly coded items (e.g., implementation obstacles). The transcript was then systematically analyzed by searching for instances of all coded items in each speaker's segment of the presentation. The data were first analyzed by presenter and then across all five presenters. All of the coded data were then further sorted and compared using Frank's (1972) framework for critiquing educational policy and reform.

3.1.2 Employing a Framework

Evaluating any educational phenomenon is a complex task and can be approached from many different theoretical perspectives or frameworks, each of which may provide a variety of analytical tools and perspectives. The benefit of such a framework, as articulated by Kubow and Fossum (2003), is that it can be used to analyze the complex factors, motivations, and ambiguities shaping educational policy and reform. It is important to note, however, that no single framework can truly encompass education's complexity and, in fact, that even the best frameworks will "limit reflective inquiry to a select number of items for analysis, thereby privileging some information and factors over others and allowing some insights to be gained while excluding from view other factors that might influence interpretation of the issue" (p. 235).

This paper uses Frank's (1972) framework, which provides a set of three criteria or elements: *policy effectiveness, theoretical adequacy,* and *empirical validity*. The *policy effectiveness* criterion requires an analysis of policies and programs so that sufficient supports have been put into place to ensure that intended outcomes are met. In the case of a computer science curriculum, for example, this would involve an examination of the fiscal and structural mechanisms that have been put in place to support curriculum development and implementation and to ensure students have access to appropriate learning/resource materials. The determination of *theoretical adequacy* can be seen as resting upon the extent to which the strategies used are sufficient and appropriate to achieve the expected outcomes. The *empirical validity* criterion can be applied to questions relating to the extent to which the evaluation of the policy initiatives are grounded in research, that is, whether scientifically proven methods have been employed to determine the success or failure of the reform.

3.1.3 Understanding Context

Before analyzing the observations of the panel participants, it is essential to place them within their

specific educational contexts. The following section therefore provides a brief description of the high school computer science curriculum in each of the five countries represented. It should be noted that while many of the countries do not identify specific years of instruction by a grade level (for example Grade 10), the panelists attempted to give what would be the U.S. grade equivalent (where possible) to make it easier to compare curricula across countries.

3.1.3.1 *Canada*

In Canada, K–12 education policy and curriculum are developed and administered at the provincial level. While many provinces have implemented some form of Information and Communications Technology curriculum, Ontario has mandated a Computer Studies curriculum consisting of *Computer Science* and *Computer Engineering* courses (Ministry of Education, 1999 & 2000). The Ontario curriculum consists of a set of eight courses beginning in Grade 9. The first course (*Integrated Technologies*) is intended as a general introduction to computing technologies, including computer applications and a small amount of programming. In Grade 10, the curriculum divides into two distinct streams: the Computer and Information Science stream and the Computer Engineering stream.

The *Computer and Information Science* stream consists of three courses to be taught in Grades 10–12. Each course must include a minimum of 110 contact hours. The course for Grade 10 introduces students to software design concepts and programming fundamentals (e.g. sequence, selection, and repetition) as well as the relationship among networks, operating systems, and applications software. The course for Grade 11 includes the study of data control and data structures, problem solving and programming, the software development cycle, and operating systems. The Grade 12 course requires students to design and implement algorithms and programs, use software development and diagnostic tools, and use file management techniques in a project setting. In addition, some of these courses contain learning expectations relating to computing ethics and careers.

The *Computer Engineering* stream consists of four courses to be taught in Grades 10–12. The single Grade 10 course provides an introduction to computer hardware and the control of external components and includes the study of logic gates, internal numbering and character representation systems, operating systems and networks, and fundamental programming concepts. Students in Grade 11 have a choice of two *Computer Engineering* courses. The Grade 11 college/university preparation course is intended for students who want to study computer science or engineering at the college or university level. This course focuses on the use of computer hardware and software to solve problems from an engineering perspective, requiring students to construct systems that use computer programs to interact with hardware and to install and configure computer hardware and software components. The Grade 11 workplace preparation course, however, is intended for students who wish to improve their practical understanding of hardware and software operations, networks, and operating systems. This course focuses on installation, maintenance, and repair procedures for computer systems and networks. The Grade 12 *Computer Engineering* course requires students to develop and construct systems and design and implement computer programs to drive real-world devices and to develop a thorough understanding of networking hardware, protocols, and configurations.

3.1.3.2 *Israel*

Unlike Canada's provincial curricula, Israel has a national curriculum for computer science. In Israel, the Ministry of Education sets all educational policy and then implements it with the aid of professional committees and supervisors. Israel's curriculum for high school computer science (Gal-Ezer, J. & Harel, D., 1999) was developed on the basis of the following

underlying principles:

- Computer Science is a full-fledged scientific subject.
- The program should concentrate on the key concepts and foundations of the field.
- The program should focus on lasting concepts of computer science, not on changing technology.
- Two different programs are needed with differing levels of requirements.
- Each of the programs should have required units and electives.
- Conceptual and experimental issues should be interwoven throughout the program.
- Two different programming paradigms should be taught to provide different ways of algorithmic thinking or different ways of solving problems.
- A well-equipped and well-maintained computer laboratory is mandatory.
- New course material must be written for all parts of the program by different teams in different academic institutions. The teams must have computer scientists on board, as well as high school teachers and researchers in computer science education.
- To be certified to teach computer science, teachers must have an adequate formal computer science education (at least an undergraduate degree in computer science).

The 450-hour curriculum is divided into five units of 90 hours each. The first three units are intended for students seeking an introduction to computer science. Students wishing to continue with their study of computer science at the university level would take two additional advanced units. The first unit (*Fundamentals 1*) covers the foundations of computer science (problem solving, algorithmic thinking, and algorithmic solutions to algorithmic problems). The second unit (*Fundamentals 2*) introduces more advanced concepts such as recursion and two-dimensional arrays. The third unit (*Applications or a Second Paradigm*) provides students with a choice of elective courses. Students intending to major in computer science can choose among courses including: *Logic Programming, Computer Organization and Assembly Language, Information Systems,* or Graphics. Students planning to complete only three units usually opt for an *Applications* course. The unit four course (*Software Design*) focuses on abstract data types and data structures. The final unit course (*Theory*), again provides students with a choice of two electives: *Computational Models or Numerical Analysis.* Although these operate as distinct courses within the curriculum, they all focus on the key concepts and foundations of computer science: algorithms, abstraction, and system design, and each of the courses that comprise the curriculum emphasize these foundations (Gal-Ezer, Beeri, Harel, & Yehudai, 1995).

3.1.3.3 *Scotland*

In Scotland, the government provides recommended curriculum guidelines for schools in all areas. While these guidelines are not mandatory, adherence is largely guaranteed through the linking of final examination content for Grades 10–12 to the recommended curriculum. Scotland's computing curriculum has been in place since 1984, with several minor revisions and a major revision in 1999. The Scottish computing curriculum for high schools (The Scottish Parliament Information Centre, 1999) consists of a flexible network of courses covering a broad range of topics. It's designed to both educate pupils about computers and also to provide application skills. At every stage of the curriculum framework, teaching, learning, and assessment follow two principles: knowledge and understanding and the application of that knowledge in a problem-solving scenario. Each course in the curriculum consists of a combination of mandatory and required units (three in total for each course). Student evaluations are drawn from a combination of coursework, course projects, and a final examination.

In each grade, students choose from a selection of courses depending upon their interests and their

performance in preceding courses. Like Israel, Scotland offers a multi-level curriculum consisting of a Standard Grade strand and a Higher strand. The Standard Grade strand consists of an Intermediate 1 *Computing Studies* course, an Intermediate 2 *Computing Studies* course, and an Intermediate 2 *Information Systems* course. The Higher strand consists of a *Higher Computing* course, a Higher *Information Systems* course, an Advanced Higher *Computing* course, and an Advanced Higher *Information Systems* course. Performance in the first Standard Grade *Computing Studies* course is assessed at three levels: foundational, general, and credit. Pupils who achieve a foundational credit would progress from the first Standard Grade *Computing Studies* course to the *Intermediate 1 Computing Studies* course. Pupils who achieve a general credit would progress from the first *Standard Grade* course to the *Intermediate 2* course. Success in this course would allow them to proceed to the *Higher* strand the following year. Students who achieve a credit grade in the first *Standard Grade* would move directly to the Higher *Computing* or *Information* systems courses. Success in these courses would then allow them to move onto an Advanced Higher course. The *Standard Grade* course is a two-year course requiring 160 instructional hours while the *Intermediate, Higher,* and *Advanced Higher* courses are one-year courses with 120 hours of instructional time.

Scotland, however, is unique among the representatives in that it provides a strand of three courses referred to as the *Access Cluster,* specifically designed for pupils who have disadvantages in learning. The *Access 1* and *2* courses concentrate on teaching life skills using computer technology. Students who succeed in the *Access 1* and *2* courses can go on to do *Access 3,* a very basic computing skills course. From this course they can go on to do *Intermediate 1* and perhaps *Intermediate 2* in Grade 12.

3.1.3.4 *South Africa*

As is the case with Israel and Scotland, educational policy in South Africa is determined at the national level, but like Ontario, it is implemented at the provincial level. Over the past 10 years, the South African National Department of Education has been involved in a complete revision of its curricula (from Grades 1–12), including its various computing curricula (Department of Education, 2005a, 2005b). The new computing curricula have been designed to be scenario-based, and so, where possible, examples are drawn from everyday life and integrated with other subject areas. As in Ontario, South Africa's computer studies curriculum consists of two distinct strands: *Computer Applications Technology and Information Technology.* The *Computer Applications Technology* strand focuses on end-user applications for Grades 10–12. The *Information Technology* strand, on the other hand focuses on the use of information and communications technologies in social and economic applications, with an emphasis on system analysis, problem solving, logical thinking, and information management and communication through the use of various software development tools (e.g. object-oriented programming). Students who opt to take the I*nformation Technology* strand will take one course per year in each of Grades 10, 11, and 12. All subjects in the new curricula to be introduced in 2006 will have 4 to 5 hours of instruction time per week with roughly 200 days (or 40 weeks) in the year available for school work. Of this tuition time, some will be used for contact time, some for practical (lab) work, and some for formal assessment (tests and examinations).

Course contents include algorithmic design, hardware and system software, networking, human computer interfacing, programming, data structures, databases and spreadsheets, testing and user interfaces, and management information systems. The *Information Technology* course is organized around four learning outcomes with the expectation that a percentage of the time in each course will be spent in each.

- Hardware and system software (20%)
- E-communications (12%)
- Social and ethical issues (8%)
- Programming and software development (60%).

As the student progresses through the courses, the content and level of complexity for each of these learning outcome areas increases appropriately until the point where, at the end of Grade 12, the students are expected to be able to design and implement a relatively complex, real-world application.

3.1.3.5 *United States*

Although there is, at present, no national curriculum for high school computer science education, the Computer Science Teacher's Association (CSTA) is actively promoting the *Model Curriculum for K–12 Computer Science Education* published by the Association for Computing Machinery (Tucker, Deek, Jones, McCowan, Stephenson, & Verno, 2003). This model curriculum is based upon the principle that students require an understanding of the place of computer science in the modern world and that the discipline of computer science interleaves both principles and skills.

The *ACM Model Curriculum* consists of four levels of courses with the expectation that all students should complete at least the first two levels. Level 1 (*Foundations of Computer Science*) is recommended for delivery at the K–8 level and consists of modules intended for use in conjunction with existing studies across the curriculum. The learning outcomes for this level are drawn primarily from the *National Educational Technology Standards* (ISTE, 2002) but include additional outcomes relating to problem solving and algorithmic thinking. Level II (*Computer Science in the Modern World*) is intended as a first computer science course for all high school students. It provides a broad overview of the discipline with the goal of preparing students for the technological world. Conceptual content includes a fundamental understanding of operating systems, networks, the Internet, problem solving, programming, careers, and issues in computing ethics. The Level III course (*Computer Science as Analysis and Design*) is a pre-Advanced Placement course in that it focuses on scientific and engineering principles. This course is intended for students interested in pursuing more advanced studies in Computer Science, Engineering, or Engineering Technology at the college or university level. Finally, the Level IV course is intended as a special projects course that would allow students to focus on a specific area of computing in which they are especially interested. It is intended to encompass a broad range of specialized courses such as AP Computer Science or courses leading to industry certification.

3.2 EXPLORING CURRICULUM DEVELOPMENT AND IMPLEMENTATION USING FRANK'S FRAMEWORK

3.2.1 Policy Effectiveness

Frank (1972) posits that evaluating the feasibility of a given educational policy or change requires consideration of three key aspects of policy effectiveness: support and dissent, monetary and other costs, and the consequences of options. Analysis of the panel transcript illustrates that the panelists from the five participating countries provide ample evidence of Frank's argument concerning the importance of these aspects as they relate to the implementation of a high school computer science curriculum.

The issue of support and dissent, for example, appears frequently in the panelists' comments, primarily in relation to discussions of the originating site of, or impetus for, the curriculum change. The panelists from Canada, Israel, and South Africa, for example, note that the computer science curriculum revision in their countries was very much a top-down process. Stephenson, for example, notes that in Ontario:

> In 1999 a new government was elected and they decided, "Let's start all over again in every subject area in the secondary or high school level. We're going to rewrite every single piece of curriculum in the province." And so courses were

> developed and implemented one grade level per year. (p. 6)

Panelists Gal-Ezer (Israel) and Chiles (South Africa) also indicate that the computer science curriculum reform in their countries were similarly conceived of and driven by the government body responsible for education. Martin, however, indicates that while the Scottish government plays a central role in the development of the national curriculum, its adoption and implementation is a matter of choice rather than mandate.

> Unlike the rest of the UK, the Scottish curriculum is not actually statutory. It is the responsibility of head teachers, but most schools do tend to follow the curriculum guidelines that were set down by our government as though they were statutory. (p. 10)

Despite the top-down decision-making evident in the majority of the represented countries, several panelists also noted their governments attempted to improve the viability of proposed educational reforms by engaging the support of key stakeholder groups at several stages of the process. Stephenson reports that in Ontario, for example, the government worked closely with the Association for Computer Studies Educators (ACSE) in the design of the new computer science curriculum. It allowed ACSE to select the team members with the goal of ensuring that the curriculum would be written by practitioners with a thorough understanding of the knowledge base of the subject area.

Stephenson further indicates that once the document was written, key stakeholder groups, including representatives from industry and parent groups, were also brought together to review and discuss the new curriculum prior to its implementation. A similar curriculum vetting process, specifically designed to address issues of both rigor and equity, was conducted in South Africa, about which Chiles notes:

> The implementation and review process involved many groups of people, including the teachers themselves, teacher unions (teacher unions are very strong in our country), the Human Rights Commission (because of the inequities of the past we had to include what was happening within human rights), and the tertiary institutions. (p. 23)

The involvement of tertiary or post-secondary institutions was also a key element in the policy implementation process in Israel, where teams of discipline specialists subjected the curriculum to a rigorous review even after its implementation, and in Scotland where universities played a key role in on-going professional development for teachers.

According to Verno, however, the situation in the United States differs markedly from that of the other countries represented on the panel. The United States is conspicuous in its lack of engagement with computing curriculum policy at both the federal and state levels. In this absence of governmental action, efforts to define a national curriculum for computer science in the United States are being driven by the Computer Science Teachers Association (CSTA), a professional membership organization representing K–12 computing teachers. In its efforts to promote a rigorous and consistent computer science curriculum, CSTA plays a prime role in the continued development and dissemination of the *ACM Model Curriculum for K–12 Computer Science Education*. It is also developing and disseminating supporting resources and teaching and learning materials, and working to improve teacher access to professional development through workshops and symposia.

As further evidence of the accuracy of Frank's framework, the discussion also demonstrates considerable panelist engagement with issues relating to funding and resource access. With the exception of the United States (where there has been no governmental support for the development of implementation of a national computer science curriculum) all of the panelists note the extent to which the goals of the curriculum reform could only be achieved with significant financial investment by the government. Gal-Ezer, for example, indicates that in Israel, "The Ministry of Education very generously budgeted the program. The budgets covered the development of the curriculum and the development

of the materials, including the salary of the developers" (p. 19). Martin notes a similar level of on-going support from the Scottish government:

> Schools were given a massive government cash injection mainly to fund multimedia technology, which was going to prove to be the most costly part of the curriculum to implement. But it was still not enough, and half way through they gave us another 80 million pounds. (pp. 21–22)

Many of the panelists, however, note that access to resources was problematic. Chiles indicates that implementation of the new computer science curriculum in South Africa has been seriously impeded by a lack of infrastructure funding, which serves to exacerbate inequities already present in the education system.

> Funding is a major problem in that the government provides no funding whatsoever to schools in terms of implementation of many of the subjects. They will provide something towards the training, but in terms of the implementation, the physical infrastructure that is required, there is no funding whatsoever. And so you can imagine what is going to happen in a situation such as we have currently, that the advantaged schools—the previously white schools that come from richer communities—will have all of the equipment; the disadvantaged schools—previously the black and Indian, colored communities—will have very little funding and will not be able to provide the equipment themselves. (p. 25)

Problems with hardware were also a factor in the initial implementation of the Scottish curriculum as, according to Martin, the Exam Board computer system required to process the results of student examinations (which are key to student placement at universities) was not installed until the day it was to be used, and then was found to have bugs, which resulted in incorrect marks or no marks at all being generated for a considerable number of students.

In both Ontario and Scotland, access to curriculum policy documents and support materials was problematic. According to Stephenson, Ontario teachers responsible for implementing the new curriculum had difficulty accessing the curriculum documents.

> The curriculum documents were all posted online and every school received just one set of documents for all courses. So if you had seven teachers in your department, they were sharing one set of curriculum [documents] … These documents were received in the schools anywhere between September 15th and September 30th, and school started September 6th. (p. 15)

While all of the panelists articulate the importance of adequate funding for teacher training as a key factor in successful curriculum implementation, the experiences of the panelists differ significantly on this issue. With the exception of the United States, all of the governments represented on the panel provided some funding for teacher training. Martin noted, for example, that teacher training on the Scottish curriculum was provided at national, regional, and local levels.

> We had documentation sent to centers for discussion and that was followed-up by national meetings with headmasters and headmistresses, and regional meetings with principle teachers of the various subjects. Some teachers were seconded to become national trainers. They went around to regional meetings training principle teachers. We did this for a year and a half, and it worked very well. The final stage of dissemination took place in the form of in-service training days. Normally these would take place within a school, but for the purpose of standardization, what happened was that, within a local authority, one school would host an in-set for all the subject teachers within that local authority, and they attended the same in-set. (p. 20)

According to Chiles, although to a much more limited extent, the government of South Africa also provided training for teachers implementing the new

Grade 10 computer science course:

> The Grade 10 training will take place across the country … So the teachers will be receiving one week's training for a course which is going to last for three years. Many of the teachers, in fact, are under-qualified and consequently have to go through a qualification phase, and the intent is to try and do that within one week. I don't think that that is a task that is going to be easily dealt with. And so we're hoping that many of the tertiary institutions ought to be offering what are called Advanced Certificates of Education. (pp. 23–24)

The government was also establishing partnerships with universities to address the issue of on-going teacher training.

Verno also highlights the role of partnerships with colleges and universities in the successful curriculum implementation of a national computer science curriculum, especially in the absence of government funding or support. In particular, she notes that CSTA funds and oversees two teacher professional development programs: Java Engagement for Teacher Training (JETT) and Teacher Engagement for Computer Science (TECS), which involve colleges and universities across the country providing workshops and on-going mentoring for local high school computer science educators.

As Kubow and Fossum (2003) indicated, the assessment of policy effectiveness in accordance with Frank's criteria also requires consideration of the policy implementation timeframe. While the panelists from Israel, Scotland, and South Africa indicate that their new computer science curricula were developed and implemented on a reasonable timeframe, according to Stephenson, the government's rush to develop and implement a new curriculum for Ontario was considerably more problematic.

> I think everyone described most accurately the whole process as ready, fire, aim. "We're going to do this new curriculum, we're going to do it right now, oh boy we better plan for it." And, so there were issues. (p. 15)

3.2.2 Theoretical Adequacy

According to Frank's critical framework, determining the theoretical adequacy of an education policy mandate requires an examination of the linkage between the outcomes desired and the strategies employed in three areas: the reasoning and justification associated with the issue, the contextualization of the rationales with respect to current and pressing needs, and the effect of tangential factors. An examination of the transcript of the international computer science panel session reveals, however, that in all cases, significantly more attention is being paid to setting forward the expectations than to determining if the strategies put into place are actually achieving those outcomes.

In both Ontario and Israel, the panelists indicate that computer science is seen as a full-fledged scientific discipline and efforts to revise the curriculum are situated in the desire to increase its rigor. In Ontario's case, Stephenson notes:

> Many of the provinces went very much toward a general IT curriculum, cutting out the rigor of their Computer Science program. But a few provinces, including Ontario have gone in a different direction. They chose to make their Computer Science curriculum both more extensive and more rigorous. We stuck to the Computer Science framework, and we looked at expanding it. (p. 6)

According to Martin, changes to the Scottish computing curriculum were similarly situated in the desire to make the program more rigorous. In this case, however, this was driven largely by University concerns regarding student preparedness for post-secondary programs. In South Africa, however, new curriculum policies required serious consideration of issues of equity and ethnicity that are at the foundation of all educational policy development and implementation in that country.

> There is a perception that Computer Studies is exclusively for the privileged white community, and so we do have a problem in terms of inclusion of the other race groups. (p. 25)

This concern with balancing rigor and equity is further evidenced in both South Africa and Scotland by additional policies that provide the opportunity for students for whom the increased rigor is problematic to take a slower, less rigorous approach to the same material. As Martin explains, this commitment to differentiated instruction is perceived as central to Scotland's commitment to providing equitable learning opportunities for all of its citizens.

> In Scotland we're very, very proud of our policy of inclusion and we firmly believe that education should be accessible to all. There is no upper leaving age to school in Scotland. We have students in our classes who are adults who have come back to school, and we also have quite a wide range of alternative assessment techniques to enable pupils with special needs to participate in national exams. (pp. 22-23)

While commitment to a more equitable system can be seen as a primary rationale for educational policies in Scotland and South Africa, in many countries these policies are also rationalized in terms of the long-term economic needs of the country. This is most clearly exemplified by Chiles of South Africa, who states:

> In terms of the expected impact of the new courses on the economy, on employment opportunities, and on future studies, we have tried as far as possible in terms of the development of the course to take into account what is happening in business and industry. (pp. 24–25)

and Verno of the United States who argues:

> I think it's time for us to make a broad commitment to CS education at the K–12 level, so our students can compete in this global environment that we are now part of. (p. 29)

The panelists' considerations of the tangential effects of new policies on education seem to be manifested most consistently in areas relating specifically to teacher preparations and qualifications. The panelists indicate many countries place insufficient impetus on the need to ensure that teachers' skills are consistent with the policies. Gal-Ezer, notes that in Israel, for example, teachers who were teaching computer science without a bachelor's degree in computer science prior to the curriculum revision were required to take 10 additional courses in order to qualify to continue teaching the subject in high schools. Stephenson also indicates that in Ontario, the new educational policies created a series of challenges for which the government had not adequately planned.

> When you change the curriculum you also get a whole lot of ancillary issues. They [the government] never passed legislation to address the teacher certification issues, or addressed the issues such as mandatory courses and the shortage of teachers. As a result, the professional association continues to shoulder the responsibility of providing the majority of opportunities for professional development to implement the curriculum for the teachers. (p. 16)

Chiles of South Africa, also poignantly illustrates that education never takes place in isolation and that the world outside can bring its own devastating tangential effects despite the best-planned reforms.

> One of the major problems that we have in education is HIV/AIDS. As you know, it's pandemic in Africa. In South Africa we have a double-edged sword in that not only are many of our teachers dying of HIV/AIDS, but many of the officials or the workforce in business and commerce are also dying, and where do they get their trained people from? Education. So we're losing teachers to HIV/AIDS on the one side directly, and we're losing teachers to industry and commerce because of HIV/AIDS on the other side. And that is a major, major problem in our country. (p. 26)

3.2.3 Empirical Validity

Frank (1972) posits that incorporating the differences between past and present plans are central to

ensuring the empirical validity of a given educational reform. Specifically, he identifies three areas that must be taken into account for this determination:

- whether the interpretations of circumstances are consistent among stakeholders,
- what signs or changes related to an issue might be relevant, and
- whether they will be broadly recognized, and the strengths and limitations of these means of judging.

The transcript of the international computer science panel indicates, however, that, among all of Frank's categories for critiquing educational policy and reform, empirical validity is the least represented in the panelists' reflections.

It is clear from Stephenson's comments that the decision to revise the computer science curriculum in Ontario was made because the provincial government had decided to revise the entire high school curriculum, and that this decision was consistent with a cyclical pattern of political decision-making rather than of broader stakeholder perceptions that change was needed.

> In 2006 all of the Computer Science courses will undergo review again. It starts all over. That has to do with the fact that the government changes every four years. So by about their second year in, it's like, "Let's really change something. Let's start with education." So the process continues. (p. 17)

While Martin (Scotland), Gal-Ezer (Israel), and Chiles (South Africa) also note that the decision to reform the high school computer science curriculum in their countries was similarly government-driven, Chiles, however, indicates that the new curriculum policies in South Africa were also driven by a broader desire among multiple stakeholder groups for educational reform that would right the wrongs of apartheid, which ended in 1994: "South Africa has in the last 10 years been through major curriculum reform. When we say major, everything that happened before 1994 has literally been dropped and is being revised" (p. 12).

For all of the panelists, the determination of whether or not the educational reform policies would achieve their intended aims and how this is to be measured seemed open to question. While both Stephenson (Canada) and Gal-Ezer (Israel) indicate that the reforms were intended to improve the scientific rigor of the curriculum, Stephenson contends that the Ontario government is focused entirely on changing how the teachers assess the students rather than on indicators of student success. Gal-Ezer (Israel) and Martin (Scotland) however, indicate that student performance on standardized tests is the marker by which their countries measure the success of the reforms.

Although educational reform policy development is clearly a top-down process in the countries represented by the panelists, the task of determining the long-term effect of these policies appears to fall more often to external bodies than to the government itself. In some cases the relationship between these bodies is a formal one, as Martin states:

> We also have a national public body called Learning and Teaching Scotland, which is seconded computing teachers and teachers from other subjects and they support the Education Department by reviewing, assessing, and developing the curriculum. (p. 22)

while in others, as Stephenson indicates, the responsibility is undertaken by an arms-length, and often educator-driven, organization:

> There was no [government] emphasis on assessing whether these courses were actually working … again, the professional association, the subject association, did that assessment to determine where the problems were. (p. 17)

From these comments it is possible to conclude that, at least in countries represented by the panel, where educational policy reform is clearly top-down, and often politically driven, surprisingly little attention is paid to determining whether or not the broad goals of the reform have actually been achieved.

3.3 CONCLUSION

Applying Frank's framework for critiquing educational policy and reform to an examination of efforts to reform the computer science curriculum in five countries provides an interesting perspective on computer science curriculum reform in specific and on curriculum reform as a whole in that it highlights areas of strength and weakness that may have long-term effects on the extent to which the reforms accomplish what they had ostensibly been designed to achieve. As this examination demonstrates, even where the reforms are top-down and politically driven, chances of successful implementation can be improved when sufficient attention is paid to policy effectiveness (especially relating to support by stakeholder groups and sufficient resource allocation), theoretical adequacy (the balance of justifications and tangential effects) and empirical validity (whether and how the long-term effects of the policy reform are measured). And while it is clear that none of the countries represented on the panel can claim an error-free policy development and reform process, each, through the honest and open examination of its successes and failures, provides beneficial insights for other countries willing to learn the lessons provided.

3.4 REFERENCES

Bureau of Labor Statistics. (2005) *U.S. Department of Labor occupational outlook handbook.* Retrieved January 12, 2005, from http://www.bls.gov/oco/ocos110.htm

Department of Education. (2005a). *Guidelines for the development of learning programmes: Computer Applications Technology.* Pretoria Department of Education, South Africa.

Department of Education. (2005b). *Guidelines for the development of learning programmes: Information Technology.* Pretoria Department of Education, South Africa.

Frank, A. G. (1972). Sociology of development and underdevelopment of society. In J. D. Cockcroft, A. G. Frank, & D. L. Johnson (Eds.). *Dependence and underdevelopment: Latin America's political economy* (pp. 321–397). Garden City, NY: Anchor.

Gal-Ezer, J., & Harel, D. (1999). Curriculum and course syllabi for high-school computer science program, *Computer Science Education, 9*(2), 114–147.

Gal-Ezer, J., Beeri C., Harel D., & Yehudai, A. (1995). A high-school program in computer science. *Computer 28*(10), 73–80.

International Society for Technology in Education (ISTE). (2002) *National educational technology standards for teachers.* Retrieved December 1, 2005, from http://www.iste.org/nets

Kubow, P. K., & Fossum, P. R. (2003). *Comparative education: Exploring issues in international context.* Upper Saddle River, NJ: Pearson Education.

Ministry of Education. (1999). *Technological education: The Ontario curriculum grades 9 and 10.* Toronto, ON: Ministry of Education, Government of Ontario, Canada.

Ministry of Education. (2000). *Technological education: The Ontario curriculum grades 11 and 12.* Toronto, ON: Ministry of Education, Government of Ontario, Canada.

National Research Council Committee on Information Technology Literacy. (1999). *Being fluent with information technology.* Washington, DC: National Academy Press.

Strauss, A. (1987). *Qualitative analysis for social scientists*. New York: Cambridge University Press.

The Scottish Parliament Information Centre. (1999). *Education in Scotland.* Retrieved June 5, 2005 from http://www.scottish.parliament.uk/business/research/pdf_subj_maps/smda-10.pdf

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C. and Verno, A. (2003). *A model curriculum for K–12 computer science: Report of the ACM K–12 Education Task Force Computer Science Curriculum Committee.* New York, NY: Association for Computing Machinery.

Verno, A., Chiles, M., Gal-Ezer, J., Martin, J., & Stephenson, C. (2005). *Finding a curriculum that fits: The United States and Beyond.* Panel presented at the National Educational Computing Conference. Philadelphia, PA.

CHAPTER FOUR

STRATEGIES FOR THE SUCCESSFUL DESIGN AND IMPLEMENTATION OF A NATIONAL HIGH SCHOOL COMPUTER SCIENCE CURRICULUM

4.0 INTRODUCTION

As the previous three chapters have shown, designing and successfully implementing a high school computer science curriculum is an exceedingly complex task involving multiple stakeholders. In the United States, this process is made even more complex by an educational system in which decision-making and funding responsibilities are dispersed and fragmented, even to the level of individual schools. This does not mean, however, that it is not possible for the United States to launch a much-needed effort to improve high school computer science education; only that doing so requires a level of understanding and commitment that encompasses leaders at every level, from the federal leaders who put forth new legislation to the state and district education administrators who interpret and ensure compliance with policy to the individual teachers who make it a classroom reality.

This chapter reviews what the previous chapters have shown, with the goal of formulating strategies and recommendations that can bring about real, fundamental, and lasting improvements to high school computer science education. These improvements will help provide students with the scientific knowledge base required to prepare them for success in the 21st century and will ensure that this country can produce a future workforce capable of the innovations that will continue to drive the global economy.

4.1 HOW DO WE DESIGN A BETTER CURRICULUM?

As demonstrated in Chapter Two, the educational research clearly shows that high school computer science education suffers because the discipline itself seems to defeat all efforts to pin it down to one, easily understood definition. This has resulted in an inconsistent self-image, an incorrect public image, and serious misconceptions among both educational decision-makers and the broader public about the scope and focus of the discipline. While in the best of all possible worlds, attempting to establish a single, universally accepted definition might seem like the most practical solution, attempting to develop such a definition would inevitably lead to more heated arguments among academics, scientists, and industry leaders. In addition, even if an agreement could be reached, the continual evolution of the theory and technology upon which it is based would render it immediately out of date.

4.1.1 Ten Core Principles

Fortunately, research shows us that it is possible to come to understand computer science and to define a curriculum for its study at the high school level that supports the understanding that computer science is a science. Here is a list of ten principles for a successful computer science curriculum.

- A high school computer science curriculum must;
- focus on the scientific principles that underlie that science;
- develop students' familiarity with the key principles of the discipline, including abstraction, complexity, modularity, and reusability;
- focus on teaching problem-solving methodologies and critical thinking skills;
- help students develop a wide range of cognitive

capabilities and practical skills, independent of specific technologies;
- be comprehensive enough to provide students with a broad overview of the field and sense of the history of the discipline so they can better appreciate how computers have been used to address real problems (It must address the breadth of the discipline and its manifestation in multiple areas of scientific engagement including hardware design, networks, graphics, databases and information retrieval, computer security, software design, programming languages, logic, programming paradigms, artificial intelligence, applications in information technology and information systems, and social issues.);
- deal explicitly with the design and analysis process so that students develop an understanding of the structure of computer systems and the processes involved in their design, development, and maintenance;
- enable students to learn and to scaffold new ideas, concepts, and skills across a series of courses that provide age-appropriate learning outcomes;
- be taught in such a way that the contents are engaging to all students regardless of gender and ethnicity;
- interweave both conceptual and experimental issues so that students come to understand both the theoretical underpinnings of the discipline and how that theory influences practice; and
- not confuse computer science with computer literacy (Teaching students how to use various software applications is not teaching computer science.).

4.1.2 Key Concepts

Like all scientific disciplines, computer science consists of a set of key concepts that are fundamental to its understanding and practice, and should therefore form the basis of any high school curriculum. The current body of research on computer science education, (as discussed in Chapter Two) has identified several concepts key to the teaching of computer science, including the following:

- **Algorithm design:** The use of a precise, step-by-step process for solving a programming problem.
- **Recursion:** A programming function or method that repeatedly calls itself until a specified condition is met. (Note while recursion was singled out in the research, it is important to note that it is just one of many concepts that could be considered under the broader category of problem solving.)
- **Abstraction and data types:** Abstraction is a process of picking out (abstracting) common features of objects and procedures. A datatype is a name or label for a set of values and some operations that one can perform on that set of values. Programming languages implicitly or explicitly support one or more datatypes. These types may act as a statically or dynamically checked constraint, ensuring valid programs for a given language.
- **Program correctness and efficiency:** Determining program correctness involves ensuring that the program runs in strict accordance with its specification. Program efficiency relates to the efficient use of resources and is generally contained in two properties: speed (the time it takes for an operation to be completed), and space (the memory or non-volatile storage used up by the construct).
- **Program testing and debugging:** Testing is a process used to help identify the correctness, completeness, and quality of developed computer software. Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware thus making it behave as expected.

The problem with these research-identified concepts, however, is that they provide an extremely limited view of the discipline of computer science. A more

comprehensive delineation of core computer science concepts, however, can be found in the National Research Council report entitled *Being Fluent with Information Technology* (National Research Council, 1999). This report puts forth a vision of Information Fluency that extends beyond the use of today's technology in one's own field (*Information Literacy*), to include the use of algorithmic thinking to solve problems and the ability to independently learn to master new technologies as they evolve. Towards this end, *Being Fluent with Information Technology* identifies 10 key concepts:

- **Computers:** Key aspects of computing, including the program as a sequence of steps, the process of program interpretation, computing memory, and peripheral devices.
- **Information systems:** The general structural features of an information system including hardware and software components, processes, and interfaces.
- **Networks:** Key attributes and aspects of information networks including their physical structure, wide-area networks and logical structure.
- **Digital representation of information:** The use of binary code.
- **Information organization:** General concepts of information organization including searching and retrieving.
- **Modeling and abstraction:** General techniques for representing real-world phenomena as computer models, for example arrays and lists of procedures.
- **Algorithmic thinking and programming:** Concepts of algorithmic thinking including repetition (iteration or recursion), data organization (records, arrays, lists), generalization, and software design processes.
- **Universality:** The understanding that any computational task can be performed by any computer.
- **Limitations of information technology:** Assessing what information technology can be applied and when it should be applied.
- **Societal impact of information and information technology:** Social concerns relating to the use of information technology such as privacy, intellectual property, security, online etiquette, hacking, and free speech.

4.1.3 Example of a Comprehensive Curriculum

It is often helpful to put these kinds of recommendations into perspective by providing an example of an actual curriculum that incorporates them. This section examines the *Model Curriculum for K–12 Computer Science Education* (Tucker, Deek, Jones, McCowan, Stephenson, & Verno, 2004) previously described in Chapter Two, with the goal of demonstrating how the ten principles and concepts outlined above can be used as the foundation for a solid curriculum framework.

- **Focus on the scientific principles that underlie that science.** The authors of the *Model Curriculum for K–12 Computer Science Education* clearly state the study of computer science is indeed a scientific endeavor and that the goal of their curriculum framework is to introduce the principles and methodology of computer science to all students.
- **Develop students' familiarity with the key principles of the discipline, including students abstraction, complexity, modularity, and reusability skills.** The *Model Curriculum for K–12 Computer Science Education* consists of four levels of learning, each of which attempts to address these key principles at an appropriate learning level for students. Both the Level II course, *Computer Science in the Modern World* and the Level III course, *Computer Science as Analysis and Design*, require students to develop an understanding of hierarchy and abstraction and to master the basic principles of software design, including modularity and reusability. The flexible framework also provides for a Level IV Topics in *Computer Science* course that could

include, among other options, an AP course that would cover modularity and reusability at an increased level of complexity or a project course that would focus on some aspect of object oriented programming and design.

- **Focus on teaching problem-solving methodologies and critical thinking skills.** All of the courses outlined in the *Model Curriculum for K–12 Computer Science Education* contain requirements for problem solving and critical thinking. Learning Level I (covering grades 1–8), for example, requires students to understand the relationship between logic and problem solving in the real world, while Level II requires students to gain a conceptual understanding of the basic steps in algorithmic problem solving. Level III also requires students to understand the limits of computing by recognizing problems that are computationally unsolvable.
- **Help students develop a wide range of cognitive capabilities and practical skills, independent of specific technologies.** The design of the *Model Curriculum for K–12 Computer Science Education* is centered in the authors' belief that students need to understand the interleaving of computer science principles and skills. Students are then expected to apply these skills (especially algorithmic thinking) in their problem-solving activities in all educational subjects. In addition, the learning outcomes specified in the curriculum are intended to ensure that students are prepared to be knowledgeable users and critics of computers as well as designers and builders of computer applications.
- **Address the breadth of the discipline and its manifestation in multiple areas of scientific engagement (hardware design, networks, graphics, databases and information retrieval, computer security, software design, programming languages, logic, programming paradigms, artificial intelligence, applications in information technology and information systems, and social issues).** Learning outcomes for all of these areas are included in all of the courses specified by the *Model Curriculum for K–12 Computer Science Education.*
- **Deal explicitly with the design and analysis.** Every course in the *Model Curriculum for K–12 Computer Science Education* contains specific learning expectations relating to problem solving, critical thinking, software design, and analysis.
- **Enable students to learn and to scaffold new ideas, concepts, and skills across a series of courses that provide age-appropriate learning outcomes.** The *Model Curriculum for K–12 Computer Science Education* builds a solid framework of conceptual and empirical learning experiences for students. By incorporating all of the National Educational Technology Standards (NETS) developed by the International Society for Technology in Education (ISTE, 2002) into its Level I section, it supports and expands the national standards for educational technology learning in grades K–8. It then provides a set of three age and learning-level appropriate courses that allow students to progress from foundational knowledge to mastery.
- **Be taught in such a way that the contents are engaging to all students regardless of gender and ethnicity.** While the *Model Curriculum for K–12 Computer Science Education* does not prescribe specific pedagogical approaches, it does highlight the importance of increasing the level of computer science knowledge for all students, and most especially for those who are members of underrepresented groups.
- **Interweave both conceptual and experimental issues so that students come to understand both the theoretical underpinnings of the discipline and how that theory influences practice.** Each high school course described in the *Model Curriculum for K–12 Computer Science Education* includes a set of conceptual learning outcomes as well as practical laboratories designed to provide students with hands-on learning.
- **Not confuse computer science with computer literacy.** Teaching students how to use various software applications is not teaching computer science. The introductory section of the *Model Curriculum for K–12 Computer Science Education* explicitly

describes the distinctions between computer science and information technology and between information literacy and information fluency.

An examination of the learning outcomes included in the *Model Curriculum for K–12 Computer Science Education* also indicates the extent to which this curriculum framework complies with the essential concepts of information fluency as set forth in the *Being Fluent with Information Technology* report. The following table illustrates coverage of these concepts in the Level II *Computer Science in the Modern World* course, and the Level III Computer Science as Analysis and Design course. (The Level I *Foundations of Computing* course is not included in this table because this is a pre-high school course. The Level IV *Topics in Computer Science* course is excluded from this table because this course can take many forms, including an AP Computer Science course, or a course geared toward professional certification.)

KEY FLUENCY CONCEPT	COURSES	LEARNING OUTCOME
Computers	Level II	Principles of computer organization and the major components (input, output, memory, storage, processing, software, operating system, etc.)
		Fundamentals of hardware design
	Level III	Hardware and systems: logic, gates and circuits, binary arithmetic, machine and assembly language, operating systems, user interfaces, compilers
Information systems	Level II	Principles of computer organization and the major components (input, output, memory, storage, processing, software, operating system, etc.)
	Level III	Design for usability
Networks	Level II	The basic components of computer networks (servers, file protection, routing protocols for connection/communication, spoolers and queues, shared resources, and fault-tolerance)
Digital representation of information	Level II	The notion of hierarchy and abstraction in computing, including high-level languages, translation (compilers, interpreters, linking), machine languages, instruction sets, and logic circuits
		The connection between elements of mathematics and computer science, including binary numbers, logic, sets, and functions
	Level III	Topics in discrete mathematics: logic, functions, sets, and their relation to computer science
Information organization	Level II	The notion of hierarchy and abstraction in computing, including high-level languages, translation (compilers, interpreters, linking), machine languages, instruction sets, and logic circuits
		The connection between elements of mathematics and computer science, including binary numbers, logic, sets, and functions
	Level III	Levels of language, software, and translation: characteristics of compilers, operating systems, and networks

Modeling and abstraction	Level II	Managing complexity through top-down and object-oriented design Procedures and parameters
	Level III	Methods (functions) and parameters Recursion Objects and classes (arrays, vectors, stacks, queues, and their uses in problem-solving) Event-driven and interactive programming
Algorithmic thinking and programming	Level II	Examples (like programming a telephone answering system) that identify the broad interdisciplinary utility of computers and algorithmic problem solving in the modern world The basic steps in algorithmic problem-solving (problem statement and exploration, examination of sample instances, design, program coding, testing and verification) Sequences, conditionals, and loops (iteration)
	Level III	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. Principles of software engineering: software projects, teams, the software life cycle
Universality	Level II	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
Limitations of information technology	Level III	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)
Societal impact of information and information technology	Level II	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 Identification of different careers in computing (e.g., information technology specialist, Web page designer, systems analyst, programmer, CIO)
	Level III	Social issues: software as intellectual property, professional practice Careers in computing: computer scientist, computer engineer, software engineer, information technologist.

While it may be one among many possible options, the *Model Curriculum for K–12 Computer Science Education* serves as an excellent example of a comprehensive and yet flexible high school computer science curriculum framework. Designed to reflect both the breadth of the discipline and the key knowledge needs of today's students, it allows schools to focus on core knowledge while at the same time incorporating local needs and existing program strengths.

4.2 HOW DO WE ENSURE SUCCESSFUL IMPLEMENTATION?

The extent to which a curriculum meets students' learning needs is a key element in decisions about what to teach. The curriculum itself, however, will have no real effects on student learning outcomes unless it is broadly and successfully implemented at the classroom level. As demonstrated in Chapter Three, however, the successful implementation of a high school computer science curriculum is a complex process requiring effort and buy-in at every level of the educational system. This section explores the key factors for effective curriculum implementation and provides practical strategies for success in three key areas: policy effectiveness, theoretical effectiveness, and empirical validity.

4.2.1 Policy Effectiveness

The sad reality is that very few educational policy initiatives effectively generate systemic change simply because there is a fundamental disconnect between those responsible for creating the policies and those responsible for making them happen. Often this gap results from a lack of understanding of the educational environment, students, and teachers. Sometimes it is the result of arrogance. It is always a precursor to failure.

Here, however, are five essential elements that will help ensure that a new or revised curriculum is translated into classroom reality in a way that truly benefits students and the broader society.

1. **Support:** The new curriculum initiative must have both top-down and bottom-up support. Whether it is driven by the federal or state government (top-down), by school district, or by teachers and parents (bottom-up) everyone must agree that change is necessary and everyone must be committed to making it happen. This also means that major change agents must be in place at all of these levels to ensure a continued level of enthusiasm and support.
2. **Stakeholder Buy-in:** External groups must have a role in the curriculum review process if they are to support the implementation process. This means that groups such as teachers unions, professional associations, parent councils, post-secondary institutions, and business and industry should be invited to take part in the review process and to help create consensus that the new curriculum is in concert with larger educational, social, and economic goals.
3. **Resources:** Schools, teachers, and students must be provided with the resources required to successfully implement the new curriculum. These resources include access to the appropriate hardware and software (it does not always have to be the latest and the greatest but it has to work) and learning resources (e.g. textbooks, reference resources, and manipulatives).
4. **Professional Development:** Teachers must receive the training necessary to allow them to master not only the content of the new curriculum but effective teaching strategies for delivering that content to students.
5. **Timeframe:** Often governments or individuals try to rush the implementation process, either because they do not understand how long it takes to achieve real systemic change, or because the appearance of doing something is more important than actually doing it right. Every step toward the successful implementation of a

new curriculum takes time. Giving less time than truly needed to accomplish any step along the path from vision to reality can condemn the entire process to failure.

4.2.2 Theoretical Effectiveness

Attempts to implement new curricula or simply improve existing ones also fail because the goals are not clear or the strategies used to achieve them are insufficient or inappropriate. As the comments of the international computer science curriculum experts in Chapter Three demonstrate, the problem often lies in the fact that the curriculum developers were attempting to achieve multiple and conflicting goals. For example, attempts to increase the academic rigor of a course or subject area may actually discourage students (especially those who might shy away from a program such as computer science) from pursuing studies in this area, thus undercutting the goal of making the curriculum more equitable. Curriculum designers must also deal with the trade-off between flexibility and the realities of school scheduling. For example, while a curriculum designed to include a broad array of possible courses may appear to offer students increased opportunities to explore their interests and abilities, in reality, it could ensure that no courses are offered simply because there are not enough students in any one course in a given school to allow that school to staff it.

Here, therefore, are three suggestions to help ensure your new or revised computer science curriculum will be theoretically effective.

1 **Be explicit:** Begin the curriculum development process with a clear statement of the intended outcomes of the curriculum.

2 **Examine all assumptions:** Do not assume that a given strategy will achieve one or more learning outcomes. Also, do not assume that because two goals are worth achieving that they will not somehow be in conflict with each other at the implementation level.

3 **Do a school-level reality check:** Ensure that those who will ultimately be responsible for implementing a new or revised curriculum have an opportunity to provide feedback on the model before the content (i.e., specific learning outcomes and strategies) is written. Once the content has been written, ask for their feedback again and make this feedback a driving force of the pre-implementation revision.

4.2.3 Empirical Validity

One of the most surprising things the research review reveals, is how little effort is put into measuring the ultimate, long-term success of a curriculum initiative. With the exception of the government of Israel, the research provides little indication that other governments have funded research to determine if a new computing curriculum for high schools has actually achieved its goals. This should not be surprising since the original curriculum models for most of the curricula that have been developed to date were not grounded in existing educational research. This lack of attention to research may explain why so few educational change initiatives achieve their stated goals and why so many teachers respond to announcements on impending changes with extreme skepticism.

Here, therefore, are three suggestions to help ensure your new or revised computer science curriculum will be empirically valid.

- **Do the research first:** Ensure that the curriculum model and content are solidly ground in the current body of international educational research. The research can be very helpful in pointing out mistakes there is no need to repeat.
- **Plan for the long-term:** Ensure that everyone involved in the project at every level of the educational system understands that systemic changes will take time and will require a long-term commitment of attention and resources.
- **Measure your success and correct when necessary:** Establish short-term and long-term metrics for success and perform appropriate

quantitative and qualitative studies to determine if they have been met. If early results show less success than hoped for, do not hesitate to make appropriate adjustments.

4.2.4 Example of a Comprehensive Implementation Plan

Recommendations regarding successful curriculum implementation can sometimes be clearer when examined in relation to an actual project that attempts to put them into practice. This section examines one particular project involving administrators, teachers, and students in the Los Angeles Unified School District (LAUSD).

The Computer Science Equity Collaborative (CSEC) was initiated by a National Science Foundation-funded research team and was formed to address disparities in representation in high school computer science along lines of race, social class, and gender. Jointly sponsored by UCLA's Graduate School of Education, UCLA's Henry Samueli School of Engineering and Applied Sciences, and the LAUSD, CSEC began with a three-year, multi-site, longitudinal study of the computer science pipeline in LAUSD high schools. This study revealed serious problems in the district's computer science education program, including the lack of a teacher computer science pipeline, a dearth of computing resources for teachers and students, and a general sense of misunderstanding of the computer science discipline by students, teachers, counselors, and administrators. This research led to the creation of a partnership focused on addressing these issues through professional development for teachers and increased support services for students studying Advanced Placement Computer Science (AP CS).

- **Support:** The partnership between the UCLA Graduate School of Education researchers, the post-secondary engineering faculty, and the LAUSD representatives ensured that teachers and students would be provided with many different kinds of support from several sources. This support included both pedagogical and technical support.
- **Stakeholder Buy-in:** The CSEC collaborative allowed for multiple roles for stakeholders that encompassed knowledge of computer science, insight on the instructional design leadership and vision, and an essential awareness of the complexities of race and gender in computing. The Dean of the School of Engineering provided overall support for diversifying the high school computer science pipeline and articulated the connections between this high school program and his own goals for a more culturally diverse enrollment in computer science at UCLA. LAUSD's Director of Science, Todd Ullah, provided passionate leadership for the project at the district level with his vision and his willingness to create change and improve computer science education in the district. John Kwan, of LAUSD's Instructional Technology Division provided much-needed planning, implementation, and coordination between the district, teachers, and schools. School administrators also provided assistance by identifying potential AP CS teachers.
- **Resources:** The Graduate School of Education provided the foundational research and contributed to all areas of curriculum and teacher support. Under the direction of Jane Margolis and Joanna Goode, researchers from the Graduate School of Education provided the facilitation between LAUSD and the Engineering School. They provided the research-based evidence to drive the reform effort and worked with both institutions to co-develop a model of change that would increase representation in computer science classes for females and students of color while also accounting for the complex ways in which schools operate. Margolis, who was the Principal Investigator of the three-year research project, carried out much of the coordinating and publicity work, including serving as lead grant writer, attending conferences, and making industry connections. Joanna Goode coordinated

and taught at the monthly AP Readiness program and served as lead instructor and planner in the professional development program. She also developed the curricular materials, coordinated with teachers via email, provided instructional materials, and administered evaluations. After the initial summer institute, follow-up research revealed that teachers found the existing district computer science curriculum to be a poor match for the needs of students in learning computer science and preparing for the AP exam. Armed with qualitative data demonstrating the inadequacy of learning resources, Goode surveyed available introductory computer science curriculum and recommended an AP-specific computer science curriculum for use by LAUSD teachers. With the approval the District, LAUSD and UCLA shared the cost of this curriculum.

The School of Engineering provided assistance with logistical issues, such as providing lab space and classrooms for high school teachers and students. A lecturer from the School of Engineering also served as a primary instructor for both the summer professional development program and monthly AP Readiness sessions for teachers and students.

LAUSD provided district-level coordination including: identifying and recruiting teachers for the program, enrolling teachers, providing substitute teachers, and providing transportation. The school district also supported the addition of AP computer science courses to schools' master schedules. District staff also took the lead for the logistical and curricular planning of the summer professional development program.

- **Professional Development:** CSEC began preparing teachers to teach the AP CS course by providing a summer program, but soon determined that one or two summer weeks was not sufficient. As a result, they then began a monthly AP Readiness program to support the teachers' ongoing knowledge and pedagogy development throughout the year. The CSEC team also worked with the teachers, rather than just trying to "train" them. They maintained flexibility in program scheduling to accommodate emerging issues, concerns, and misunderstandings. They also made a point of acknowledging many of the teachers in the room as experts—respecting them while also providing a safe space for new AP CS teachers to wrestle with computer science concepts. Finally, they helped build leadership within the teaching community by allowing teachers to take on new roles in helping other teachers while improving their own teaching practices.
- **Timeframe:** The project's three-year research program exposed many of the underlying issues of access, equity, and the rigor of computer science in LA schools and provided a plan for how to address them. Pending new funding, the researchers predict that the project will be completed in 2009.

Several additional elements of the program helped to ensure that it met the criteria for empirical validity.

- **Do the research first:** All of the interventions proposed in this program followed from the extensive three-year study preceding project implementation. The project also involved specialists from several disciplines to ensure that several different kinds of expertise would be brought to help interpret the research and to solve the problems that the data uncovered.
- **Plan for the long-term:** All of the partners in the project were committed to maintaining the implementation plan. Project organizers admit, however, that they have concerns about the sustainability of this effort and the pressure of outside forces that have the potential to marginalize the importance of computer science. They note that the national educational dialogue (and policy) now is not about access, rigor, deep thinking, scientific reasoning, or inquiry. Rather, it is about failing schools, a narrow focus on literacy and numeracy (as determined by testable topics) accountability, and so on. In this climate. it is difficult to

maintain computer science as a vital part of the K–12 educational system.

- **Measure your success and correct when necessary:** The CSEC team is using several data points to evaluate the success of the project. They are collecting and analyzing data on annual computer science enrollment numbers (e.g., how has enrollment changed) by overall courses, overall enrollment, and also by gender and race. They also collected data at each meeting with teachers and students to examine the change in their understandings and practices over time. They continue to conduct research to determine teacher professional development and involvement, and if the programs are sustainable after the research grants expire.

The CSEC project was designed to ensure sustainability by involving multiple levels of stakeholder buy-in and support. Unlike many efforts to improve K–12 computing education, this project provides a relevant example of a change initiative that is solidly grounded in research (a three-year study) and clear in its focus and objectives (to improve equity of access by traditionally underrepresented students of color and females to computer science education by increasing the opportunities for students to take the AP CS course). The CSEC project leaders were also careful to acknowledge and include teachers, to recognize and support their craft knowledge, to encourage and develop their leadership, and to support their ongoing development as exemplary educators.

4.3 HOW DO WE IMPROVE HOW TEACHERS TEACH?

One of the challenges decision-makers face when attempting to improve existing curricula or introduce new ones is determining the extent to which they can and should also attempt to shape how teachers teach. Pedagogy, however, is a complex phenomenon, both science and art, and as such it has proven almost impossible to quantify or replicate.

4.3.1 Five Qualities of Exemplary Teachers

Although we do not currently have a guaranteed process for ensuring that all teachers are great teachers, research into computer science education has shown that truly effective teachers demonstrate several key qualities that help them engage and teach students more effectively. Here are five key qualities of exemplary high school computer science teachers.

- **Problem-solving approach:** Exemplary computer science teachers use a problem-solving approach that allows students to examine problems from different angles and perspectives and formulate solutions.
- **Real-world focus:** Exemplary computer science teachers motivate students by having them create real-world artifacts with an intended audience and encouraging them to understand the essential link between the problem, the user, and the solution
- **Explicit emphasis on design:** Exemplary computer science teachers explicitly teach and use the software design process, ensuring that students master the steps involved in designing, creating, testing, and debugging software.
- **A welcoming environment:** Exemplary computer science teachers make their classroom a welcoming environment for all students (especially young women and minority students) and find creative ways to engage all students with examples and exercises that are relevant to their lives.
- **Modeling life-long learning:** Exemplary computer science teachers serve as role models for their students by continuing to enhance their own teaching and technology skills and by exploring new ideas and new technologies.

4.3.2 Seven Systemic Changes Required to Improve Teaching

For teachers, the challenge of becoming and remaining exemplary educators is often exacerbated

by a system of pre-service education and teacher certification that makes it so difficult to become a computer science teacher that it actually discourages highly qualified individuals from applying in the first place. In many states, there is no pre-service program that specifically prepares high school computer science teachers. In many school districts, there are no core knowledge guidelines for computer science teachers. In many schools, teachers are expected to take on teaching responsibilities in subject areas for which they have little or no preparation (Computer Science Teachers Association, 2005).

The responsibility for ensuring that all high school computer science teachers are knowledgeable, well prepared, and continuously engaged must be shared at all levels of the education system. It requires a profound commitment, a coordinated set of policies, and the allocation of appropriate and adequate resources. Here is a list of seven policy changes that would substantially improve computer science education in the United States.

- **Mastery of core knowledge:** High school computer science teachers should be required to have completed an undergraduate degree in computer science or a comparable degree program.
- **Standardized pre-service programs:** All teacher preparation programs should be required to adhere to the National Council for Accreditation of Teacher Education (NCATE) standards for high school computer science educators.
- **Certification standards:** State teacher certification requirements for high school computer science teachers should adhere to a consistent (and enforced) national standard that would allow for greater clarity and mobility from state to state.
- **Professional development:** School districts should provide regular professional development for computer science teachers to allow them to keep their knowledge and skills current.
- **Focus on teaching:** School districts should employ a sufficient number of technical specialists with responsibility to ensure that computer hardware, networks, and software is maintained, freeing teachers to concentrate on their teaching.
- **Competitive compensation:** Salaries for computer science teachers should be commensurate with those offered in industry to ensure that the best possible candidates prepare and apply for teaching positions.
- **Professional affiliation:** All high school, computer science teachers should be members of professional associations that support their discipline-based knowledge and provide a teaching community that mentors and celebrates them.

4.4 CALL TO ACTION

Computer science has increasingly become a core knowledge requirement for all educated citizens. Like the more traditional sciences, it provides an essential understanding of the world around us. Even if we are not aware of it, computers are part of almost every aspect of our lives (e.g., banking, health care, shopping, and travel) and it is vital that we understand their capabilities and their limitations. All students need a basic level of knowledge about how to use computers safely and securely. Computer science also teaches students how to be innovative and solve problems and, in this way, helps them be better prepared for college and careers.

There is also a broader national interest at stake. U.S. government labor forecasts clearly indicate that we are not producing enough computer science graduates to meet the needs of industry or to compete in an increasingly global economy (Bureau of Labor Statistics, 2005). In addition, fewer college students are enrolling in computer science courses, and fewer graduates with computer science degrees are going on to earn their PhDs (Taulbee, 2003). If we do not begin to address these pipeline issues immediately and at the earliest possible educational level, our ability to compete in the global economy and to participate in the great scientific and industrial developments of this new century will continue to diminish.

Addressing these issues is not just a school issue, it is an issue that can only be addressed with vision,

action, and commitment at all levels of the political and educational systems. The following sections provide suggestions for ways in which we can work together to achieve long-term, systemic improvements to computer science education. These include federal government policy makers, state government policy makers, school district policy makers, school principals, teachers, university and college faculty, and business and industry leaders.

4.4.1 Federal Government Policy Makers

Here are 10 ways federal-level policy makers and leaders can improve high school computer science education:

- Talk about the importance of computer science when you are speaking at events and with other federal leaders.
- Include computer science in the 21st century skills conversation. When people talk about the importance of Science, Technology, Engineering and Math (commonly referred to as "STEM" skills), emphasize and clarify the "T." Making the most of technology education is not just wiring schools or teaching students to use computers. That is only half the battle.
- Insist that funded STEM initiatives include requirements to address issues in K–12 computer science education.
- Support new initiatives that provide funding for research that will help improve computer science education.
- Support new initiatives that provide funding for computer science teacher training.
- Put forward or support an initiative that encourages states to rationalize their computer science teacher certification requirements. This will encourage the development of a larger, stronger, and better-qualified educational workforce.
- Put forward or support an initiative that encourages states to ensure that computer science teachers have computer science qualifications.
- Make sure conversations about the competitiveness of the United States encourage students to go into computer science careers and explain why it is so important for our nation to be a leader in this field.
- Talk to people in your community about outsourcing technology jobs. Explain how this trend can be combated by focusing on the education of future workers in our own country.
- Help improve the public's understanding of the many exciting, varied, and cutting edge jobs available in the field of computer science.

4.4.2 State Government Policy Makers

Here are 10 ways state-level policy makers and leaders can improve high school computer science education:

- Ask what local school authorities are doing to ensure students are learning what they need to get the jobs of the future and require all high school students to take at least one computer science course.
- Identify how much is being spent on computer science education in your state and determine if there have been cutbacks on computer science education.
- Promote funding for computer science course equipment and materials.
- Ensure that schools of education and on-the-job professional development opportunities are adequately training computer science teachers in your area.
- Ensure that certifications requirements for computer science teachers guarantee that teachers have the appropriate knowledge base to teach in this discipline.
- Provide programs for individuals transitioning from the information technology industry to education that offer an opportunity to acquire the educational knowledge and training needed to become an exemplary teacher.

- Talk to your local leaders and community members about what they see as future computer science job needs.
- Encourage businesses to provide support and mentoring opportunities for schools. Explain the importance of business' role in the education of future workers.
- Talk to people in your community about outsourcing technology jobs. Explain how this trend can be combated by focusing on the education of future workers in your state.
- Point community members to the resources of nonprofit groups, such as CSTA, that provide curriculum models and other resources to support computer science teachers.

4.4.3 School District Policy Makers

Here are 10 ways school district policy makers, curriculum policy leaders, and instructional technology specialists can improve high school computer science education:

- Identify how much is being spent on computer science education in your school district and determine if there have been cutbacks on computer science education.
- Promote funding for computer science courses, equipment, and materials.
- Put policies in place to ensure that only qualified teachers are teaching computer science courses and enforce those policies.
- Review your current computer science curriculum to determine whether it provides sufficient opportunities for students to gain the knowledge and skills they need to succeed in an increasingly computerized world.
- Review your current computer science curriculum to determine whether it is both academically rigorous and welcoming to all students.
- Create and support professional development opportunities for computer science teachers to ensure that their technical and pedagogical skills keep pace with change.
- Help teachers keep their focus on teaching by providing technical support staff who are responsible for maintaining and upgrading the hardware and software.
- Provide opportunities for computer science teachers across the district to meet with and mentor each other.
- Work with school counselors to ensure that they are providing students with accurate and appropriate information about career opportunities in computing and the educational pathways students must take to achieve their long-term goals.
- Point community members to the resources of nonprofit groups, such as CSTA, that provide curriculum models and other resources to support computer science teachers.

4.4.4 School Principals

Here are 10 ways school principals can improve high school computer science education:

- Identify how much is being spent on computer science education in your school and determine if there have been cutbacks on computer science education.
- Promote funding for computer science courses, equipment, and materials.
- Ensure that only qualified teachers are teaching computer science courses.
- Review your current computer science curriculum to determine whether it provides sufficient opportunities for students to gain the knowledge and skills they need to succeed in an increasingly computerized world.
- Work with your computer science teachers to ensure that their courses are both academically rigorous and welcoming to all students.
- Create and support professional development opportunities that help computer science teachers ensure that their technical and pedagogical skills keep pace with student learning needs.

- Help teachers keep their focus on teaching by providing technical support staff who are responsible for maintaining and upgrading both the hardware and software.
- Provide opportunities for computer science teachers to meet with colleagues from across the district, the state, and the nation.
- Work with school counselors to ensure they are providing students with accurate and appropriate information about career opportunities in computing and the educational pathways students must take to achieve their long-term goals.
- Point teachers to the resources of nonprofit groups, such as CSTA, that provide curriculum models and other resources to support computer science teachers.

4.4.5 Teachers

Here are 10 ways teachers can improve high school computer science education:

- Review your current computer science curriculum to determine whether it provides sufficient opportunities for students to gain the knowledge and skills they need to succeed in an increasingly computerized world.
- Ensure that your courses are both academically rigorous and welcoming to all students.
- Model life-long learning to your students by seeking out and taking advantage of relevant professional development opportunities.
- Lobby for funding for computer science courses, equipment, and materials.
- Lobby for your program by speaking to parent groups, other teachers, and individual students about computer science.
- Help dispel the myth that there are no job opportunities in computer science. There are many exciting and cutting-edge jobs available.
- Work with school counselors to ensure that they are providing your students with accurate and appropriate information about career opportunities in computing and the educational pathways students must take to achieve their long-term goals.
- Create opportunities to meet with computer science teachers from across the district so that you can share resources and strategies.
- Become a member of a professional association, such as CSTA, that provides curriculum models, support materials, mentoring, and community and that helps you continue to develop your teaching and leadership skills.

4.4.6 University and College Faculty

Here are 10 ways university and college faculty in computing and engineering programs and in schools of education can improve high school computer science education:

- Support high school computer science education by requiring students entering your university to have taken at least one high school computer science course.
- Create different entry points into your programs for students with differing levels of high school computing experience.
- Provide local high school teachers and students with opportunities to interact with your faculty, graduate students, and undergraduate students by inviting them to visit your institution or offering to send faculty and students to speak at local schools.
- Provide teachers and students with detailed information about your computer science and engineering programs and the skills incoming students require to ensure their success at your institution.
- Create opportunities for your female and minority students to mentor high school students whose life experiences are similar to their own.
- Create and support professional development opportunities that help computer science teachers ensure that their technical and

pedagogical skills keep pace with change.
- Provide teachers, students, and school counselors with information about careers in computer science and the higher education pathways required for students to meet their long-term goals.
- Ensure that schools of education are adequately preparing pre-service teachers to teach computer science.
- Help teachers purchase new technology resources by working with them to develop grant applications.
- Ensure that your faculty are aware of the *Model Curriculum for K–12 Computer Science Education* and that they play an active role in professional organizations such as CSTA that support computer science teachers.

4.4.7 Business and Industry Leaders

Here are 10 ways business and industry leaders can improve high school computer science education:

- Make sure conversations about the competitiveness of the United States encourage students to go into computer science careers and explain why it is so important for our nation to be a leader in this field.
- Help improve the public's understanding of the many exciting, varied, and cutting edge jobs available in the field of computer science.
- Include computer science in the 21st century skills conversation. When people talk about the importance of Science, Technology, Engineering and Math (commonly referred to as "STEM" skills), emphasize and clarify the "T." Making the most of technology education is not just wiring schools or teaching students to use computer applications. That is only half the battle.
- Talk to people in your community about outsourcing technology jobs. Explain how this trend can be combated by focusing on the education of future workers in our own country.
- Build partnerships with schools that promote and demonstrate excellence in computer science education.
- Support the national implementation of curriculum standards such as the *ACM Model Curriculum for K–12 Computer Science and the National Educational Technology Standards.*
- Require that all computer education projects receiving funding from your organization demonstrate consistency with these national standards.
- Fund new research initiatives that will help improve computer science education.
- Fund new teacher professional development initiatives for computer science teachers.
- Support an initiative that encourages states to ensure that computer science teachers have computer science qualifications.

4.5 CONCLUSION

Our goal for this chapter was to provide practical solutions for a complex problem, and while we cannot claim that any of our lists are exhaustive, they do provide a powerful set of suggestions that, if put into practice, will significantly improve computer science education in high schools. The reality is that we are at somewhat of a crossroads. We know that the world continues to change and that many of the changes we face are related to the burgeoning possibilities and consequences of our growing dependence upon and interrelation with computing technology. Maintaining our ability to meet the challenges of the present and future require us to think very carefully about the kind of knowledge our students need to grow and to succeed. Supporting and improving high school computer science education and ensuring that the opportunities it provides are open to all students requires a multi-level commitment. It is a commitment we must make if our schools are to continue to provide relevant education and our society is to continue to solve problems on the cutting edge of innovation.

4.6 REFERENCES

Bureau of Labor Statistics. (2005) *U.S. Department of Labor Occupational Outlook Handbook.* Retrieved January 12, 2005, from http://www.bls.gov/oco/ocos110.htm

Computer Science Teachers Association. (2005). Results of the national secondary computer science survey. Retrieved October 16, 2005, from http://csta.acm.org/Research/sub/CSTANationalSurvey2004.html

International Society for Technology in Education (ISTE). (2002) *National educational technology standards for students.* Retrieved December 1, 2005, from http://www.iste.org/nets

National Research Council Committee on Information Technology Literacy. (1999). *Being fluent with information technology.* Washington, DC: National Academy Press.

Taulbee, O. E. (2003). 2002-2003 *Taulbee study: Undergraduate enrollments drop; Department growth expectations moderate.* Retrieved December 22, 2005, from http://www.cra.org/statistics/

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C. and Verno, A. (2003). *A Model curriculum for K–12 computer science: Report of the ACM K–12 Education Task Force Computer Science Curriculum Committee.* New York, NY: Association for Computing Machinery

BIBLIOGRAPHY

ACM K–12 Task Force. (2005). *ACM K–12 Task Force Curriculum Survey Results.* Retrieved December 22, 2005, from http://csta.acm.org/Research/sub/TuckerSurveyResults.html

Adajian, L. B. (1996). Professional communities: Teachers supporting teachers. *Mathematics Teacher, 89*(4), 321–364.

Almstrum, V. L., Hazzan, O., & Ginat D. (Eds.). (2004). Special issue on import/export relationships to computer science education research. *Computer Science Education, 14*(4).

Armoni, M., & Gal-Ezer, J. (2003). Non-determinism in CS high school curricula. *Proceedings of the FIE 2003 Conference,* Boulder, CO, F2C–18 – F2C–23.

Ballantyne, R., McLean, S. V., & Macpherson, I. (2003). *Knowledge and skills required for creating a culture of innovation: Supporting innovative teaching and learning practices.* Brisbane: Faculty of Education, Queensland University of Technology.

Bauch, P. A., & Goldring, E. B. (2000). Teacher work context and parent involvement in urban high schools of choice. *Educational Research and Evaluation, 6*(1), 1–23.

Bell, D. (1983). Promoting business education through teacher participation in professional associations. *Business Education Forum/Yearbook, 37*(8), 48–51.

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching, 20*(1), 45–73.

Ben-Ari, M. (2002). From theory to experiment to practice in CS education. Paper presented at *Kolin Kolistelut—Koli Calling: 2nd Annual Finnish/Baltic Sea Conference on Computer Science Education,* Koli, Finland, October 2002.

Ben-Ari, M. (2004). Situated learning in computer science education, *Computer Science Education, 14*(2), 85–100.

Ben-David Kolikant, Y. (2004). Learning concurrency: Evolution of students' understanding of synchronization. *The International Journal of Human Computers Studies, 60*(2), 243–268.

Ben-David Kolikant, Y., & Pollack, S. (2004a). Establishing computer science professional norms among high school students. *Computer Science Education, 14*(1), 21–35.

Ben-David Kolikant, Y., & Pollack, S. (2004b). Community-oriented pedagogy for inservice CS teacher training. *Proceedings of the ITiCSE 2004 Conference,* Leeds, United Kingdom, 191–195.

Ben-David Kolikant, Y. (2005). Students' alternative standards for correctness. P*roceedings of the First International Computing Education Research Workshop,* October 1–2, 2005, University of Washington, Seattle, WA, 37–43.

Bergin, S., & Reilly, R. (2005). Programming: Factors that influence success. *Proceedings of the SIGCSE 2005 Conference,* St. Louis, Missouri, USA, 411–415.

Böszörmenyi, L. (2005). Teaching: People to people—about people: A plea for the historic and human view. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 93–103.

Bureau of Labor Statistics. (2005) *U.S. Department of Labor occupational outlook handbook.* Retrieved January 12, 2005, from http://www.bls.gov/oco/ocos110.htm

Campbell, P. F., & McCabe, G. P. (1984). Predicting the success of freshmen in a computer science major. *Communications of the ACM, 27*(11), 1108–1113.

Cantwell Wilson, B., & Shrock, S. (2001). Contributing to success in an introductory computer science course: A study of twelve factors. *SIGCSE Bulletin, 33*(1), 184–188.

Cantwell Wilson, B. (2002). A study of factors promoting success in computer science including gender differences. *Computer Science Education, 12*(1–2), 141–164.

Cartelli, A. C. (2002). Computer science education in Italy: A survey. *SIGCSE Bulletin, 34*(4), 36–39.

Collofello, J. S. (2002). Creation, development and evaluation of an innovative secondary school software development curriculum module. *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference,* Boston, MA.

Computer Science Teachers Association. (2005). *Results of the national secondary computer science survey*. Retrieved October 16, 2005, from http://csta.acm.org/Research/sub/CSTANationalSurvey2004.html

Computer Science Teachers Association (2005a). *Strategic Plan.* Retrieved December 22, 2005, from http://csta.acm.org/About/sub/CSTAStrategicPlan.html/

Crombie, G., Abarbanel, T., & Anderson, C. (2000). All-female computer science: The positive effects of single gender classes. *The Science Teacher,* March 2000, 40–43.

Dagienë, V. (2005). Teaching information technology in general education: Challenges and perspectives. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 53–64.

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1988). *Draft report of the ACM Task Force on the Core of Computer Science.* New York, NY: Association for Computing Machinery.

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a discipline. *Communications of the ACM, 32*(1), 9–23.

Denning, P. J. (2004). Great principles in computing curricula. *Proceedings of the SIGCSE 2004 Conference,* Norfolk, Virginia, USA, 336–341.

Department of Education. (2005a). *Guidelines for the development of learning programmes: Computer Applications Technology.* Pretoria Department of Education, South Africa.

Department of Education. (2005b). *Guidelines for the development of learning programmes: Information Technology.* Pretoria Department of Education, South Africa.

Dorninger, C. (2005). Educational standards in school informatics in Austria. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary* Schools—Evolution and Perspectives, ISSEP 2005, Klagenfurt, Austria, March/April 2005, 66–69.

du Boulay, B., O'Shea, T., & Monk, J. (1989). The black box inside the glass box: Presenting computing concepts to novices. In E. Soloway & J. C. Spohrer (Eds.). *Studying the novice programmer* (pp. 431–446).

Hillsdale, NJ: Lawrence Erlbaum Associates.

Driscoll, M. (1987). *Ten case studies from a study of exemplary mathematics programs.* Reston, VA: National Council of Teachers of Mathematics.

Eurydice (Information Network on Education in Europe). (2005). *Information and communication technologies in education.* Retrieved January 25, 2005 from http:// www.eurydice.org/Documents/ Bibliographie/en/frameset_biblio_refs_en.html

Fadi, P. D., & McHugh, J. A. (2000). Problem-solving methodologies and the development of critical thinking skills. *Journal of Computer Science Education, 14*(1&2), 6–12.

Fadi, P. D., & McHugh, J. A. (2001). Prototype software development tools for beginning programming. *Journal of Computer Science Education, 14*(3&4), 14–20.

Felleisen, M., Findler, R. B., Flatt, M., & Krishnamurthi, S. (2002). The structure and interpretation of the computer science curriculum. *Journal of Functional Programming, 14*(4), 365–378.

Fincher, S. (1999). What are we doing when we teach programming? *Proceedings of the FIE1999 Conference,* San Juan, Puerto Rico, 12a4–1–12a4–4.

Fincher, S., & Petre, M. (2004). *Computer science education research.* London, UK: Routledge Falmer.

Fisher, A., & Margolis, J. (2002). Unlocking the clubhouse: The Carnegie Mellon experience. *SIGCSE Bulletin, 34*(2), 79–83.

Foley, J. (2004). Old challenges, new opportunities. *Computing Research News, 16*(4). Retrieved December 1, 2005 from http:// www.cra.org/ CRN/articles/sept04/foley.html

Frank, A. G. (1972). Sociology of development and underdevelopment of society. In J. D. Cockcroft, A. G. Frank, & D. L. Johnson (Eds.). *Dependence and underdevelopment: Latin America's political economy* (pp. 321–397). Garden City, NY: Anchor.

Franklin, R. (1987). What academic impact are high school computing courses having on the entry-level computer science curriculum? P*roceedings of the SIGCSE 1987 Conference,* St. Louis, Missouri, USA, 253–256.

Fullan, M. (2002). The change leader. *Educational Leadership, 59*(8), 16–21.

Gal-Ezer, J., Beeri, C., Harel, D., & Yehudai, A. (1995). A high school program in computer science. *Computer, 28*(10), 73–80.

Gal-Ezer, J., & Harel, D. (1999). Curriculum and course syllabi for high-school computer science program, *Computer Science Education, 9*(2), 114–147.

Gal-Ezer, J., & Harel, D. (1998). What (else) should CS educators know? *Communications of the ACM, 41*(9), 77–84.

Gal-Ezer, J., & Zeldes, A. (2000). Teaching software designing skills. *Computer Science Education, 10*(1), 25–38.

Gal-Ezer, J., & Zur, E. (2002). The concept of "algorithm efficiency" in the high school CS curriculum. *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference,* Boston, MA, T2C1–T2C6.

Gal-Ezer, J., Vilner, T., & Zur, E. (2003). *Characteristics of students who failed (or succeeded) the introductory CS course.* Paper presented at the *FIEE 2003 Conference.* Boulder, CO. Retrieved December 4, 2005 from http:// fie.engrng.pitt.edu/fie2003

Ginat, D. (1996). Efficiency of algorithms for programming beginners. *Proceedings of the SIGCSE 1996 Conference.* Philadelphia, PA, 256–260.

Ginat, D. (2000). Colorful examples for elaborating exploration of regularities in high school CS1. *Proceedings of ITiCSE 2000,* Helsinki, Finland, 81–84.

Goldweber, M., Fincher, S., Clark, M., & Pears. A. (2004). The relationships between CS education research and the SIGCSE community. *SIGCSE Bulletin, 36*(1), 147–148.

Graham, S., & Latulipe, C. (2003). CS girls rock: Sparking interest in computer science and debunking the stereotypes, *Proceedings of the SIGCSE 2003 Conference,* Reno, Nevada, USA, 322–326.

Grandell, L. (2005). High school students learning university level computer science on the web—a case study of the DASK-model. *Journal of Information Technology Education, 4*, 207–218.

Greening, T. (1998). Computer Science: Through the eyes of potential students, *Proceedings of the ACSE 1998 Conference,* Brisbane, Australia, 145–154.

Gries, D. (2002). Where is programming methodology these days? *SIGCSE Bulletin, 34*(4), 5–7.

Gurbiel, E., Hardt-Olejniczak, G., & Kolczyk, E. (2005). Informatics and ICT in the Polish education system. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 93–103.

Gupta, U. G., & Houtz, L. E. (2000). High school students' perceptions of information technology skills and careers. *Journal of Industrial Technology, 16*(4), 2–8.

Guzdial, M. and Soloway, E. (2003). Computer science is more important than calculus: The challenge of living up to our potential. *SIGCSE Bulletin, 35*(2), 5–8.

Haberman, B., & Ginat, D. (1999). Distance learning model with local workshop sessions applied to in-service teacher training, *Proceedings of the ITiCSE 1999 Conference,* Krakow, Poland, 64–67.

Haberman, B., & Ben-David Kolikant, Y. (2001). Activating 'black boxes' instead of opening 'zippers'—A method of teaching novices basic CS concepts. *Proceedings of the ITiCSE 2001 Conference,* Dublin, Ireland, 41–44.

Haberman, B., & Averbuch, H. (2002). The case of base cases: Why are they so difficult to recognize? Student difficulties with recursion. *Proceedings of the ITiCSE 2002 Conference,* Aarhus, Denmark, 84–88.

Haberman, B., Shapiro, E., & Scherz, Z. (2002). Are black boxes transparent?—High school students' strategies of using abstract data types. *Journal of Educational Computing Research, 27*(4), 411–436.

Haberman, B., Lev, L., & Langley, D. (2003). Action research as a tool for promoting teacher awareness of students' conceptual understanding. *Proceedings of the ITiCSE 2003 Conference,* Thessaloniki, Greece, 144–148.

Haberman, B. (2004a). High school students' attitudes regarding procedural abstraction: Special issue devoted to recent research projects of secondary informatics education. *Education and Information Technologies, 9*(2), 131–145.

Haberman, B. (2004b). How learning logic programming affects recursion comprehension. *Computer Science Education, 14*(1), 37–53.

Haberman, B., Averbuch, H., &. Ginat, D. (2005). Is it really an algorithm?—The need for explicit discourse. *Proceedings of the ITiCSE 2005 Conference,* Lisbon, Portugal, 74–78.

Hagan, D., & Markham, S. (2000). Does it help to have some programming experience before

beginning a computing degree program? *SIGCSE Bulletin, 32*(3), 25–28.

Harris, J. W. (1987). Teacher and leader—why we must be both. *Vocational Education Journal, 62*(5), 26–28.

Hazzan O., Impagliazzo, J., Lister, R., & Schocken, S. (2005). Using history of computing to address problems and opportunities, *Proceedings of the SIGCSE 2005 Conference,* St. Louis, MO, 126–127.

Hazzan, O., & Lapidot, T. (2004a). Construction of a professional perception in the "Methods of Teaching Computer Science" course. *SIGCSE Bulletin, 36*(2), 57–61.

Hazzan, O., & Lapidot, T. (2004b). The practicum in computer science education: Bridging the gap between theoretical knowledge and actual performance. *SIGCSE Bulletin, 36*(4), 47–51.

Holmboe, C., McIver, L., & George, C. (2001). Research agenda for computer science education. *Proceedings of the 13th Workshop of the Psychology of Programming Interest Group,* Bournemouth, UK, 207–233.

Impagliazzo, J., & Lee, J. A. N. (2004). Using computing history to enhance teaching. *Proceedings of the History of Computing in Education 2004 Conference,* Toulouse, France, 165–176.

Information Technology Association of America. (2002). *Bouncing Back: Job skills and the continuing demand for IT workers.* Arlington, VA: Information Technology Association of America.

Inos, R. H., & Quigley, M. A. (1995). *Synthesis of the research on educational change. Part 4: The teacher's role.* Honolulu, HI: Pacific Region Educational Laboratory.

International Society for Technology in Education (ISTE). (2002) *National educational technology standards for students.* Retrieved December 1, 2005, from http://www.iste.org/nets

International Society for Technology in Education (ISTE). (2002) *National educational technology standards for teachers.* Retrieved December 1, 2005, from http://www.iste.org/nets

Jeffrey, N. (1996). The responsibility of professional associations. Paper presented at *A Meeting of the Minds: ITEC Virtual Conference 1996,* Australia.

Kavander, T., & Salakoski, T. (2004). Where have all the flowers gone? Computer science education in general upper secondary school. *Proceedings of the Kolin Kolistelut—Koli Calling Conference,* Koli, Finland, 112–115.

Kubow, P. K., & Fossum, P. R. (2003). *Comparative education: Exploring issues in international context.* Upper Saddle River, NJ: Pearson Education.

Kuznetsov, A. A., & Beshenkov, S. A. (2005). Russian educational standards of informatics and informatics technologies (ICT): Aims, content, perspectives. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 70–73.

Lapidot, T., & Hazzan, O. (2003). Methods of teaching computer science courses for prospective teachers. *SIGCSE Bulletin, 35*(4), 29–34.

Latulipe, C., & Graham, S. (2005). Degaming the curriculum: A failed experiment. Retrieved December 10, 2005, from: http://hci.uwaterloo.ca/students/clatulip/research.html

Lee, P. (2004). *The computer science brain drain: A call to revitalize computer science education.* Retrieved April 23, 2004, from https://www.wiki.cs.cmu.edu/public/uploads/Main/it-talent.pdf

Levy, D., & Lapidot, T. (2002). Shared terminology, private syntax: The case of recursive descriptions.

Proceedings of the ITiCSE 2002 Conference, Aarhus, Denmark, 89–93.

Loidl, S., Muhlbacher, J., & Scauer, H. (2005). Preparatory knowledge: Propaedeutic in informatics. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 93–103.

McLaughlin, M. W. (1992). How district communities do and do not foster teacher pride. *Educational Leadership, 50*(1), 33–35.

Merritt, S. (1995). Reflections of a computer scientist for teachers and teacher educators. In J. D. Tinsley and T. J. van Weert (Eds.). *Proceedings of the World Conference on Computers in Education* VI (4790486). London: Chapman and Hall for the International Federation of Information Processing.

Micheuz, P. (2005). 20 years of computers and informatics in Austria's secondary academic schools. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 20–31.

Ministry of Education. (1999). *Technological education: The Ontario curriculum grades 9 and 10.* Toronto, ON: Ministry of Education, Government of Ontario, Canada.

Ministry of Education. (2000). *Technological education: The Ontario curriculum grades 11 and 12.* Toronto, ON: Ministry of Education, Government of Ontario, Canada.

Mitchell, W. (2002). Information technology education: One state's experience. *Journal of Computing in Small Colleges, 17*(4), 123–132.

Mittermeir, R. T. (Ed.) (2005). *From computer literacy to informatics fundamentals. International Conference on Informatics in Secondary Schools-—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005.

Moorman, P., & Johnson, E. (2003). Still a stranger here: Attitudes among secondary school students towards computer science. *Proceedings of the ITiCSE 2003 Conference,* Thessaloniki, Greece, 193–197.

Morris, J. H. and Lee, P. (2004). The incredibly shrinking pipeline is not just for women anymore. *Computing Research News, 16*(1), 20.

Muller, O., Haberman, B., & Averbuch, H. (2004). (An almost) pedagogical pattern for pattern-based problem-solving instruction. *Proceedings of the ITiCSE 2004 Conference,* Leeds, United Kingdom, 102–106.

Muller, O. (2005). Pattern oriented instruction and the enhancement of analogical reasoning. *Proceedings of The First International Computing Education Research Workshop,* October 1–2, 2005. Seattle, WA, 57–67.

National Research Council Committee on Information Technology Literacy. (1999). *Being fluent with information technology.* Washington, DC: National Academy Press.

O'Lander, R. (1996). Factors effecting high school student's choice of computer science as a major. *Proceedings of the Symposium on Computers and the Quality of Life,* Philadelphia, Pennsylvania, USA, 25–31.

Paz, T., & Lapidot, T. (2004). Emergence of automated assignment conceptions in a functional programming course. *Proceedings of the ITiCSE 2004 Conference,* Leeds, United Kingdom, 181–185.

Payton, F. C. (2003). Rethinking the digital divide. *Communications of the ACM, 46*(6), 89–91.

Peckham, J., DiPippo, L., Reynolds, J., Paris, J., Monte, P., & Constantinidis, P.A. (2000), First course in computer science: The discipline is more than programming. *Journal of Computing in Small Colleges, 15*(5), 223–230.

Perkins, D., Schwartz, S., & Simmons, R. (1988). Instructional strategies for the problems of novice programmers. In R. E. Mayer (Ed.), *Teaching and learning computer programming* (pp. 153–178). Hillsdale, NJ: Lawrence Erlbaum Associates.

Pham, B. (1997). The changing curriculum of computing and information technology in Australia. *Proceedings of the ACSE 1997 Conference,* Melbourne, Australia, 149–154.

Poirot, J. L. (1979). Computer education in the secondary school: Problems and solutions. *Proceedings of the SIGCSE 1979 Conference,* Dayton, Ohio, 101–104.

Poirot, J. L., Taylor, H. G., & Norris, C. A. (1988). Retraining teachers to teach high school computer science. *Communications of the ACM, 37*(7), 912–917.

Pollack, S., & Scherz, Z. (2005). Supporting project development in CS – the effect on intrinsic and extrinsic motivation. *Proceedings of the 10th PEG Conference,* Tampere, Finland, 143–148.

Ramberg, P. (1986). A new look at an old problem: Keys to success for computer science students. *ACM SIGCSE Bulletin, 18*(3), 36–39.

Ragonis, N., & Ben-Ari, M. (2005). A long-term investigation of the comprehension of OOP concepts by novices. *Computer Science Education, 15*(3), 203–221.

Rich, L., Perry, H., & Guzdial, M. (2004). A CS1 course designed to address interests of women. *Proceedings of the SIGCSE 2004 Conference,* Norfolk, Virginia, USA, 190–194.

Reiter, A. (2005). Incorporation of informatics in Austrian education: The project "Computer-Education-Society" in the school year 1984/85. In R. T. Mittermeir (Ed.) *From computer literacy to informatics fundamentals, Proceedings of International Conference on Informatics in Secondary Schools—Evolution and Perspectives, ISSEP 2005,* Klagenfurt, Austria, March/April 2005, 4–19.

Roberts, E. (2004). The dream of a common language: The search for simplicity and stability in computer science education. *Proceedings of the SIGCSE 2004 Conference,* Norfolk, Virginia, USA, 115–119.

Roberts, E,. & Halopoff, G. (2005). Computer Science Teachers Association analysis of high school survey data. Retrieved December 2, 2005 from http://csta.acm.org/Research/sub/CSTANational Survey2004.html

Romberg, T. A., & Middleton, J. A. (1995). Conceptions of mathematics and mathematics education held by teachers. In N. Webb & T. A. Romberg (Eds.). *Collaboration as a process for reform.* New York: Teachers College Press.

Rountree, N., Vilner, T., Wilson, B., & Boyle, R. (2004). Predictors for success in studying CS, panel session. *Proceedings of the SIGCSE 2004 Conference,* Norfolk, Virginia, USA, 145–146.

Rosenthal, T., Suppes, P., & Ben-Zvi, N. (2002). Automated evaluation methods with attention to individual differences—A study of a computer-based course in C, *Proceedings of the ASEE/IEEE Frontiers in Education Conference,* Boston, MA.

Sabin, M., Higgs, B., Riabov, V., & Moreira, A. (2005). Designing a pre-college computing course. *Journal of Computing in Small Colleges, 20*(5), 123–132.

Sargent, J. (2004). An overview of past and projected employment changes in the professional IT occupations. *Computing Research News, 16*(3), 1–22.

Scherz, Z., & Haberman, B. (2005). Mini-projects development in computer science—Students' use of organization tools. *Informatics in Education, 4,* 307–319.

Schollmeyer, M. (1996). Computer programming in high school vs. college. *Proceedings of the SIGCSE 1996 Conference,* Philadelphia, PA, USA, 378–382.

Shackelford, R. (2005). *Why can't smart people figure out what to do about computing education.* Presentation at the CSTA/ISTE Computer Science and Information Technology Symposium, Philadelphia, PA.

Sims-Knight, J. E., & Upchurch, R. L. (1993). Teaching software design: A new approach to high school computer science. Paper presented at *The Annual Meeting of the American Educational Research Association,* Atlanta, GA. Retrieved November 1, 2005 from http://www2.umassd.edu/cisw3/people/faculty/rupchurch/

Sleeman, D., Putnam, R. T., Baxter, J. A, & Kuspa, L. (1989). A summary of misconceptions of high school basic programmers. In E. Soloway & J. C. Spohrer (Eds.). *Studying the novice programmer* (pp. 301–314). Hillsdale, NJ: Lawrence Erlbaum Associates.

Stephenson, C. (2002). High school computer science education. *Journal of Computer Science Education,* Annual, 30–40.

Stevenson, D. E. (1993). Science, computational science and computer science: At a crossroads. *Proceedings of the 1993 ACM Conference on Computer Science,* Indianapolis, IN, 7–14.

Stiller, E., & LeBlanc, C. (2003). Creating new computer science curricula for the new millennium. *Journal of Computing in Small Colleges, 18*(5), 198–209.

Strauss, A. (1987). *Qualitative analysis for social scientists.* New York: Cambridge University Press.

Taulbee, O. E. (2003). *2002–2003 Taulbee study: Undergraduate enrollments drop; Department growth expectations moderate.* Retrieved December 22, 2005, from http://www.cra.org/statistics/

Taylor, H. G., & Mounfield, L. C. (1989). The effect of high school computer science, gender, and work on success in college computer science. *ACM SIGCSE Bulletin, 21*(1), 195–198.

Taylor, H. G., & Mounfield, L. C. (1991). An analysis of success factors in college computer science: High school methodology is a key element. *Journal of Research on Computing in Education, 24*(2), 240–245.

The College Board. (2005). *AP report to the nation.* Retrieved December 22, 2005, from http://www.collegeboard.com/about/news_info/ap/2005/index.html

The Scottish Parliament Information Centre. (1999). *Education in Scotland.* Retrieved June 5, 2005 from http://www.scottish.parliament.uk/business/research/pdf_subj_maps/smda-10.pdf

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C. and Verno, A. (2003). *A model curriculum for K–12 computer science: Report of the ACM K–12 Education Task Force Computer Science Curriculum Committee.* New York, NY: Association for Computing Machinery.

Vandenberg, S., & Wollowski, M. (2000). Introducing computer science using a breadth-first approach and functional programming. *Proceedings of the SIGCSE 2000 Conference,* Austin, TX, USA, 180–184.

Ventura, P., & Ramamurthy, B. (2004). Wanted: CS1 students. No experience required. *Proceedings of*

the SIGCSE 2004 Conference, Norfolk, Virginia, USA, 240–244.

Verno, A., Chiles, M., Gal-Ezer, J., Martin, J., & Stephenson, C. (2005). *Finding a curriculum that fits: The United States and Beyond.* Panel presented at the National Educational Computing Conference. Philadelphia, PA.

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,
- 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: 800.342.6626
Email: cstahelp@acm.org

The Following Companies are Gold Level Sponsors of CSTA

IBM Microsoft Sun microsystems

NOTES

NOTES