

Computer Science Education

Looking Back and Looking Ahead



Countess of Lovelace wrote first program in 1800.



Herman Hollerith used punch card system in 1890.



Grace Hopper helped invent COBOL in 1961.



Seymour Papert developed Logo in the early 1960s.



Niklaus Wirth invented Pascal in the 1970s.



Who's next? One of your students perhaps?

By Chris Stephenson

Subject: Computer science

Grade Level: 9–16 (Ages 14 & up)

Audience: Teachers, technology coordinators, teacher educators, library/media specialists

Standards: NETS•S 1; NETS•T 1 (www.iste.org/standards)

“... the real excitement of the discipline lies in the fact that computer science and computer science education have been built on imagination and discovery that goes back to our earliest times.”

In what seems like a relatively short time, computers have become part of our everyday lives. We bank by computer, we shop by computer, and we even initiate and support many of our personal and professional relationships by computer. It is thus somewhat surprising that so few people truly understand how computers work or pause to think about the remarkable changes this technology is causing for us and our society.

This lack of understanding can have serious consequences, because as is the case in so many areas of our lives, knowledge is power. Students who do not understand the technology cannot fully appreciate the many opportunities it provides for study and for lifelong career choices. On a broader scale, countries whose citizens don't understand the technology will be at an economic disadvantage because they will be forced to simply use rather than build the tools needed by the rest of the world.

The mass media, always quick to project our best hopes and worst fears, has created images as diverse as the vacuuming robots of the *Jetsons* cartoon, the dangerously self-aware HAL of *2001: A Space Odyssey*, and the Pinocchio-like cyber-creations of *A.I.: Artificial Intelligence*. And though these images have captured our imaginations and sparked our curiosity, they have never really encompassed the complexity and beauty of this science in a way that engages young people and encourages them to take a serious interest.

This job has fallen to the computer teacher. Besides communicating the fundamental concepts of the discipline to their students, they are responsible for representing the scope of a vast and rapidly evolving scientific discipline and, more importantly, for encouraging

students to pursue their own knowledge so that society's long-term needs for a tech-savvy work force will be met.

A History of Ideas and Innovation

One of the problems with computer science is that it is often perceived as machine-focussed and isolating. When many students think about computing, they imagine people who spend all their time sitting in front of machines. This perspective has too often been supported by educators who focus on one programming assignment after another rather than giving the students a sense of the history of imagination and discovery that has always been essential to computing. Teachers who understand and can communicate this sense of excitement not only engage students more fully but also provide them with a more realistic context in which to judge both the benefits and drawbacks of new technologies.

For some people, the history of computer science and so of computer science education is a history of machines, and certainly that aspect of the discipline has been marked by seemingly unbounded progress. Computing machines here in North America began with the punch-card technology of Herman Hollerith in 1890. They continued with the gargantuan computing machines such as the Mark 1, ENIAC, EDVAC, and the IBM 701 and 650. The assistance of first vacuum tubes, then transistors, and finally integrated circuits made possible the revolution of the personal computer and the introduction of the Altair 8800, Apple II, and IBM PC in the late 1970s and early 1980s.

For me, however, the real excitement of the discipline lies in the fact that computer science and computer science education have been built on imagina-

tion and discovery that goes back to our earliest times. Like most sciences, ours is built on foundations established by ancient civilizations. The Greeks, for example, developed axiomatic mathematics and formal logic, while the Egyptians developed multiplication tables, tables for squares and roots, exponential tables, and the formula for quadratic equations. In the late 16th and early 17th centuries, the development of algebra, logarithms, the slide rule, and analytic geometry contributed greatly to our ability to perform complex mathematical operations.

Perhaps more important, however, the history of computer science has also been a history of people capable of marrying current needs with new ideas and visions. Wilhelm Schickard designed and built what is believed to be the first digital calculator in 1623. Charles Babbage envisioned two steam-powered computational machines called the Analytic Engine and the Difference Engine that could perform mathematical computations to eight decimal places. Herman Hollerith invented a new punch-card technology, which helped tabulate the results of the 1890 U.S. Census. He went on to set up the Tabulating Machines Company that later became IBM. Alan Turing helped the British government break the Enigma code during World War II, providing a decisive advantage to the Allied forces. More recently, Steve Jobs and Steve Wozniak envisioned an easy-to-use computer for the masses and went on to build Apple, while Bill Gates launched the operating system (DOS) that built Microsoft.

Computer Programming

When many of today's students sit down at the computer to write a program, they have no idea how the tech-

Unlike universities, which face intense pressure to provide students with skills that will make them immediately employable in an ever-changing industry, high school educators operate in a time frame that allows them to concentrate on foundational concepts and skills.

nology that allows them to do so was developed. Students who do have a sense of programming history, however, are more likely to understand how many idiosyncrasies of today's programming practices have roots in the technological limitations of the past.

The history of computer programming is also about ideas put into action to achieve innovation. As computers have evolved from early prototypes to massive machines to laptop and handheld tools, the ways in which information and instructions are prepared for computers has also changed radically. As a result, both what and how we teach students, and our expectations of what they can accomplish with the tools at hand, have undergone constant revision.

Ada, Countess of Lovelace and daughter of the poet Lord Byron, developed the first set of instructions for Babbage's Analytic Engine, in effect the first computer program, in 1800. From that time until 1958, computers were used by a very select group of people primarily for mathematical and scientific calculation. In 1958, however, J. N. Patterson Hume and C. C. Gotlieb published the first book on using computers for business (*High-Speed Data Processing*) and so pioneered today's wide-scale use of computing technology.

The major problem with programming in the early days was that it was time-consuming and difficult work resulting primarily from the need for the programmer to communicate in machine language, which consisted entirely of 1s and 0s. A major improvement in programming languages began in the 1950s with the development of

symbolic machine languages, which were collectively referred to as assembly language. Assembly language allowed programmers to write instructions using letter symbols rather than binary operation codes.

As more businesses began to appreciate the potential for computers, new languages were developed to meet particular needs. The development of high-level programming languages, which included their own translation software to translate the programmer's instructions into machine code, opened the doors of computer science education. In 1957, an IBM-sponsored committee headed by John Backus introduced a high-level programming language called FORTRAN (Formula Translator). In 1961, Grace Murray Hopper helped invent COBOL (Common Business-Oriented Language) for processing business data. This language helped revolutionize the banking and insurance industries and is still used and studied today.

This process continues unabated, as newer, more general-purpose programming languages—Ada, C, C++, LISP, Java, and Visual Basic (to name just a few)—continue to be introduced and adopted by industry.

Programming Languages for Education

The increasing importance of computing in business was echoed in education, especially at universities. The need to train students to master the complexities of computer programming led to the development of new programming languages that were particularly effective for teaching fundamental computing concepts. These languages,

when combined with the increasing accessibility of computers to education provided by the introduction of personal or desktop computers, helped bring computer science into the high school classroom.

In 1964, John Kemeny and Thomas Kurtz of Dartmouth invented BASIC (Beginners All-Purpose Instruction Code) to help undergraduate students learn programming more easily. At the same time, Seymour Papert at the Massachusetts Institute of Technology was developing Logo to help much younger students explore a mathematical environment that used an on-screen "turtle" to draw and create simple animation.

Perhaps no programming language had as much widespread impact on education, however, as Pascal. Developed by Niklaus Wirth of Switzerland's Federal Institute of Technology and named after mathematician Blaise Pascal, Pascal provided a major benefit over BASIC in that it was designed to support the concepts of structured programming.

Programming Paradigms

As computer programs and the tasks they performed became larger and more complex, scientists and educators had to find new ways of thinking about, developing, and maintaining these programs over time. In other words, the changing demands of programming and the evolution of computer hardware have given rise to a number of programming paradigms.

Structured programming is one such paradigm. At its heart is the idea that a computer program should be structured so that its execution is easy to follow, not hopelessly tangled like a plate of spaghetti. In structured programs, groups of statements are used to control the flow of information. Control constructs such as linear sequence, repetition, and selection help programmers to organize their programs to the extent that they can be read from top to bottom.

Structured programming also concerns how programs are designed and developed. Today when educators teach about structured programming, they are not simply referring to the control structures but to top-down programming. Top-down programming is a systematic way of analyzing computer problems by breaking them down into a series of problems, each of which is solved. Once all the smaller problems are solved, the solutions are combined to solve the larger problem.

The move from the unstructured to the structured paradigm provided a significant improvement in how programs were written and developed. By following the rules associated with structured programming, professional programmers and students could make the logic of their programs easier to follow. This meant that it was also easier for them to find and fix errors and change their programs as needed. But the reality is that even structured programs are often time consuming to write, hard to understand, and inflexible in that they can only be used for a single task. (This is one of the main reasons software is often expensive and does not always work as well as it should.)

As computers became more ubiquitous and programs larger and more complex, the need to produce software more efficiently gave rise to the object-oriented programming paradigm. Object-oriented programming is a method of designing and writing programs based on the concept that a program is a collection of objects that work together.

Object-oriented programming is based on three main ideas. The first is that software should be divided into parts, called objects, which have details hidden in them. Each of these objects has three aspects:

- what it is,
- what it does, and
- what it is called.

The second idea is that an object is created from a class from which many such objects can be created. The third is that one class can inherit (or borrow) features from another class.

Combining these three ideas allows programmers to write large programs that are more manageable. Each object can be changed without affecting other objects. Classes that perform specific tasks can also be reused in other programs, thus reducing program development time.

Classroom Realities

All of these innovations and new concepts have had profound effects on education. From the earliest days of programming with punch cards and spaghetti programs to the current push to object-oriented programming languages such as Java, teachers have walked a thin line between technology and pedagogy, striving to determine what computer science students need to know and how best to teach them.

Unlike universities, which face intense pressure to provide students with skills that will make them immediately employable in an ever-changing industry, high school educators operate in a time frame that allows them to concentrate on foundational concepts and skills. Unlike many of their university colleagues, however, most high school educators have struggled to do so in an environment of constant shortage.

Resources For many teachers, simply accessing sufficient hardware and software is a challenge. Schools have limited budgets, and with the push to integrate computer use across the curriculum, computer science has in many cases become the poor cousin of educational computing. This is despite the fact that high school computer science

Schools have limited budgets, and with the push to integrate computer use across the curriculum, computer science has in many cases become the poor cousin of educational computing.

teachers are often expected to do the lion's share of the work of maintaining the school's computing facilities, including managing the network, loading new software, mentoring other teachers, and even fixing the school secretary's printer.

Curriculum. Determining curriculum content is also a challenge. Despite the efforts of organizations such as the Association for Computing Machinery (ACM), which developed and published a Model High School Computer Science Curriculum in 1991, curriculum content in computer science varies enormously from state to state, district to district, and even school to school. (*Editor's note:* Find the ACM's model curriculum and other URLs under Resources at the end of the article.)

In fact, many teachers will tell you the only standards they have are the requirements for Advanced Placement (AP) Computer Science, and these courses were never intended to be standard curricula. As a result, many teachers say there is no "computer science for the rest of the students."

Professional Development. Recent changes in the AP requirements have highlighted another important issue in high school computer science: the lack of ongoing professional development for teachers. As mentioned previously, computer science is fraught with changes—changing technology and changing paradigms. It is interesting to note, for example, that when most teachers were in university, object-oriented programming did not even exist. Now, if they want to teach the AP courses, they not only have to learn C++ and soon Java themselves, they also have to learn how to teach it to their students.

Computer Science continued on page 44.

Computer Science continued from page 9.

ISTE is making efforts to provide professional development through its yearly Computer Science and Information Technology Symposium. This event provides high school computer science and information technology teachers with a full day of hands-on learning in conjunction with NECC. Though this is a wonderful and important event, the number of teachers that can be accommodated in no way fills the need for localized, ongoing, curriculum-centered teacher learning.

Gender Equity. The other major issue that hangs over computer science is the dwindling number of young women entering the discipline with the goal of working in the industry. Over the past few years there has been growing concern with the fact that demand for workers in the computing industry continually outstrips availability. A 2001 study of the IT work force by the Information Technology Association of America, for example, found a deficit of more than 400,000 workers in 2001. In an era when many traditional occupations in areas such as resources and manufacturing are disappearing, computing remains one field where students with the proper skills and educational experiences can achieve a viable and rewarding career. At the same time, even though women have made major contributions to the field from its very beginnings, it continues to be a career that many young women avoid.

As Dr. Tracy Camp (1997) points out, the number of young women in computer science has continued to fall since its high in 1985. And as Carolyn Boyce (1999) notes, research indicates that this trend is linked to both institu-

tionalized discrimination and an internalized lack of self-confidence.

In many ways, the gender equity problem in high school computer science is even more pressing than at the university level. The fact that we are dealing with younger students gives some hope that we still have time to give them a more positive attitude toward the discipline and help them gain a more realistic view of their own abilities and accomplishments. The problem is that there is still a lack of training for teachers in equity issues. Some teachers have no idea how to make their program more inclusive and more interesting to all students. Others feel frustrated by the fact that the countless studies and excellent one-off projects never seem to produce widely implementable solutions that can be adapted to fit local classrooms and curricula.

The real concern here is that teachers may just give up on gender equity altogether. This would leave a U.S. industry begging for highly skilled workers with no option. With half the population missing from the equation, there is no hope of a solution.

What Education Needs

This sounds like a great deal of gloom and doom and perhaps belies the fact that there are many wonderful things occurring in high school computer science classes everywhere. Teachers are working hard to keep their skills current with changing technologies and ideas and to make their classrooms more inclusive, engaging, and relevant to their students. And students continue to challenge themselves to learn and to apply their knowledge in interesting and, one hopes, beneficial ways.

In an era when many traditional occupations in areas such as resources and manufacturing are disappearing, computing remains one field where students with the proper skills and educational experiences can achieve a viable and rewarding career.

Accomplishing our potential, however, requires a serious reengagement at the highest levels with issues of teacher training and professional development, curriculum standardization, and resource allocation. Teachers need a better idea of what they should be teaching and how best to teach it. They need opportunities to continue to expand their own knowledge and skills, and they need the proper time and resources to meet their students' needs. It is not beyond reasonable bounds to say that our future depends on these things.

First Steps. Meeting these challenges is no simple task, but steps can be taken at the state and national levels to begin to greatly improve computer science education.

- Establishing a national, or at least statewide, standard curriculum for high school computer science that includes both introductory and AP courses would provide guidance for teachers and guarantee a more consistent learning experience for students.
- Incentive-based hiring would help attract candidates with strong computing skills to the teaching profession.
- States should make long-term (and appropriately funded) commitments to ongoing professional development for teachers. This training must be classroom-focused, relevant to individual learning needs, and mandatory.
- Teachers need to have more opportunities to meet to discuss their discipline. This can be supported through funding and release time to attend state and national conferences by educational computing organizations.

Despite temporary fallbacks, the high tech sector remains a key contributor to the financial health of the United States. A strong computer science program at the high school level is the first step along the path to ensuring that this industry remains competitive enough to play a leading role among nations. Innovation grounded in good education is always a wise investment.

Teachers need a better idea of what they should be teaching and how best to teach it. They need opportunities to continue to expand their own knowledge and skills, and they need the proper time and resources to meet their students' needs.

Resources

Textbooks

- Blohm, H., Beer, S., & Suzuki, D. (1986). *From pebbles to computer, the thread*. Toronto, ON, Canada: Oxford University Press.
- Hiltzik, M. (1989). *Dealers of lightning, Xerox Parc and the dawn of the computer age*. New York: Harper Collins Publishers Inc.
- Hodges, A. (1983). *Alan Turing, the enigma*. London: Vintage.
- Hume, J. N. P., & Stephenson, C. (2000). *Introduction to programming in Java*. Toronto, ON, Canada: Holt Software Associates.

Web Sites

- ACM Model High School Computer Science Curriculum: www.acm.org/education/hscur/
- Advanced Placement Computer Science: www.collegeboard.com/ap/students/compsci/
- The Computer History Museum: <http://virtualmuseum.dlib.vt.edu/>
- Computer History through the Internet Web Quest: <http://coe.west.asu.edu/students/gdelph/Webquest.htm>
- The History of Computing (Virginia Tech): <http://ei.cs.vt.edu/~history/>
- An Internet Treasure Hunt on the History of Computing: www.kn.pacbell.com/wired/fil/pages/hunthistory1.html
- ISTE's Computer Science and Information Technology Symposium: www.iste.org/profdev/symposia/index.html#COMP_SCI
- The Machine that Changed the World: <http://ei.cs.vt.edu/~history/TMTCTW.html>
- The Women of ENIAC: www.gecdsb.on.ca/d&g/women/women.htm

References

- Boyce, C. (1999). *Attitudes of women in computer science: 1991–1999* [Online]. Available: www.bluepoof.com/Colloquium/attitudes.html
- Hume, J. N. P., & Gotlieb, C. C. (1958). *High-speed data processing*. New York: McGraw-Hill.
- Camp, T. (1997). *The incredible shrinking pipeline* [Online preprint]. Available: www.mines.edu/fs_home/tcamp/cacm/paper.html
- Information Technology Association of America. (2001). *Bridging the gap: Information technology skills for a new millennium* [Executive summary]. Arlington, VA: Author. Available: www.ita.org/workforce/studies/hw00execsumm.htm

Chris Stephenson is a research associate at the University of Waterloo and president of Holt Software Associates. She received her BA from the University of Toronto, a Bachelor of Journalism from Carleton University, and a MEd from the Ontario Institute for Studies in Education. Her research is shared between computer science and telecommunications. Chris is a past president of the International Society for Technology in Education's Computer Science Special Interest Group (SIGCS) and serves as SIGCS officer. She has written for numerous educational publications and edited and written a number of high school textbooks. She also chairs ACM's K–12 Education Task Force. Chris first became enamored with computers when she worked as a lowly production assistant in educational television in the early 1980s. Though she still has great appreciation for computers as tools to enhance both science and art, she wishes more effort were being made to make them more accessible, more intuitive, and more compatible (to each other and actual human beings).

Do you have a fascinating tidbit of information about the history of computer science education? Perhaps a vision of its future? L&L readers want to hear about it. Send a letter to L&L editor, Kate Conley, at letters@iste.org.



Evoke the spirit of a field trip without the permission slips, lost lunches, and expense. Use the classroom strategies for virtual field trips found in our new V-Trip Travel Guide!

Order online at
www.iste.org/bookstore

ISTE Customer Service
480 Charnelton Street
Eugene, OR 97401-2626 USA

Phone
800.336.5191 (U.S. & Canada)
541.302.3777 (International)
Fax 541.302.3778
E-mail orders@iste.org
Web www.iste.org/bookstore


iste