The Legacy of Computer Science

Gerald Jay Sussman
Matsushita Professor of Electrical Engineering
Massachusetts Institute of Technology

We have witnessed and participated in great advances, in transportation, in computation, in communication, and in biotechnology. But the advances that look like giant steps to us will pale into insignificance by contrast to the even bigger steps in the future. Sometimes I try to imagine what we, the technologists of the second half of the 20th century, will be remembered for, if anything, hundreds of years from now.

In the distant past there were people who lived on the banks of the Nile River. Each year the Nile overflowed its banks, wiping out land boundaries but providing fertile soil for growing crops. As a matter of economic necessity the Egyptians invented ways of surveying the land. They also invented ways of measuring time, to help predict the yearly deluge. Similar discoveries were made in many places in the world. Holders of this practical knowledge were held in high esteem and the knowlege was transferred to future generations through secret cults. These early surveyors laid the foundation for the development of geometry ("earth measurement" in Greek) by Pythagoras and Euclid and their colleagues around 350 B.C. Geometry is a precise language for talking about space. It can be taught to children. (Euclid's *Elements* has been used in this way for over two thousand years.) It makes the children smarter, by giving them ways of expressing knowledge about arrangements in space and time. It is because of these Greeks that we can tell a child, "If you build it out of triangles it will not collapse the way it does when you build it out of rectangles."

The Rhind Papyrus from Egypt (c. 1650 B.C.) is the earliest document that we have that discusses what we now think of as algebra problems. Diophantus, another Greek, wrote a book about these ideas in the 3rd century A.D. Algebra was further developed by Abu Abd-Allah ibn Musa al-Khwarizmi (c. 780 – c. 850) and others. (Note: "algebra = al'jabr" is an Arabic word meaning "the recombining of broken parts.") Algebra is also a precise language, which gives us the ability to express knowledge about the relationships among quantities, and to make deductions from that knowledge, without necessarily knowing the values of those quantities.

For a long time people were able to predict the motions of some of the heavenly bodies using *ad hoc* theories derived from observation and philosophical considerations. Claudius Ptolemy wrote the *Almagest*, a famous

compendium of this knowledge, in the second century. About 350 years ago Descartes, Galileo, Newton, Leibnitz, Euler, and their contemporaries turned mechanics into a formal science. In the process they invented continuous variables, coordinate geometry, and calculus. We now can talk about motion precisely. This achievement gives us the words to say such sentences as, "When the car struck the tree it was going 50 km/hour." Now every child can understand this sentence and know what is meant by it.

In each of these cases there was an advance in human intelligence, ultimately available to ordinary children, that was precipitated by an advance in mathematics, the precise means of expression. Such advances are preceded by a long history of informal development of practical technique. We are now in the midst of an intellectual revolution that promises to have as much impact on human culture as the cases I have just described.

The 19th century was the century of applied classical mechanics: Humans invented ways of using stored chemical energy to drive the machines of industry. The consequential development of effective means of transportation, such as railroads and steamships, transformed the society. This development continues into the 20th century with automobiles and airplanes, but the main thrust of technology in the 20th century has been applied atomic physics (as electronics). This century has been spectacular in the invention of means of communication and means of manipulation of information. We have seen the development of telecommunications, broadcast media, and most recently the Internet. We are now nearly at the end of the electronics revolution, and we are passing into the age of information and applied biology.

In particular, biological cells are self-reproducing chemical factories that are controlled by a program written in the genetic code. Current progress in biology will soon provide us with an understanding of how the code of existing organisms produces their characteristic structure and behavior. As engineers we can take control of this process by inventing codes (and more importantly, automated means for aiding the understanding, construction, and debugging of such codes) to make novel organisms with particular desired properties. Employing information technology, the future holds promise for the development of means to control biological processes that are just as effective as our current control of electrical physics.

Besides the obvious application of the control of biological processes to medicine, we will be able to coopt biological processes to manufacture novel materials and structures at a molecular scale. Of course, one of the most important products of mass-produced molecular-scale engineering will be

extremely compact, efficient, and effective computing mechanisms. Indeed, cellular computing opens a new frontier of engineering that we expect to dominate the technology of the next century.

We have been programming universal computers for about 50 years. The practice of computation arose from military, scientific, business, and accounting applications. Just as the early Egyptian surveyors probably thought of themselves as experts in the development and application of surveying instruments, we have developed a priestly cult of "computer scientists." But, as I have pointed out,[1]

> ... "Computer Science" is not a science, and its ultimate significance has little to do with computers. The computer revolution is a revolution in the way we think and in the way we express what we think. The essence of this change is the emergence of what might best be called *procedural epistemology*—the study of the structure of knowledge from an imperative point of view, as opposed to the more declarative point of view taken by classical mathematical subjects. Traditional mathematics provides a framework for dealing precisely with notions of "what is." Computation provides a framework for dealing precisely with notions of "how to."

Computation provides us with new tools to express ourselves. This has already had an impact on the way we teach other engineering subjects. For example, one often hears a student or teacher complain that the student knows the "theory" of the material but cannot effectively solve problems. We should not be surprised: the student has no formal way to learn technique. We expect the student to learn to solve problems by an inefficient process: the student watches the teacher solve a few problems, hoping to abstract the general procedures from the teacher's behavior on particular examples. The student is never given any instructions on how to abstract from examples, nor is the student given any language for expressing what has been learned. It is hard to learn what one cannot express.

In particular, in an introductory subject on electrical circuits we show students the mathematical descriptions of the behaviors of idealized circuit elements such as resistors, capacitors, inductors, diodes, and transistors. We also show them the formulation of Kirchoff's laws, which describe the

[1] from Abelson, Sussman, and Sussman, *Structure and Interpretation of Computer Programs*, 2nd Edition, MIT Press, 1996.

behaviors of interconnections. From these facts it is possible, in principle, to deduce the behavior of an interconnected combination of components. However, it is not easy to teach the techniques of circuit analysis. The problem is that for most interesting circuits there are many equations and the equations are quite complicated. So it takes organizational skills and judgment to formulate the useful equations effectively and to deduce the interesting behaviors from those equations.

Traditionally, we try to communicate these skills by carefully solving selected problems on a blackboard, explaining our reasoning and organization. We hope that the students can learn by emulation, from our examples. However, the process of induction of a general plan from specific examples does not work very well, so it takes many examples and much hard work on the part of the faculty and students to transfer the skills.

However, if I can assume that my students are literate in a computer programming language, then I can use programs to communicate ideas about *how to* solve problems: I can write programs that describe the general technique of solving a class of problems and give that program to the students to read. Such a program is precise and unambiguous—it can be executed by a dumb computer! In a nicely designed computer language a well written program can be read by students, who will then have a precise description of the general method, to guide their understanding. With a readable program and a few well-chosen examples it is much easier to learn the skills. Such intellectual skills are very hard to transfer without the medium of computer programming. Indeed,[2]

> a computer language is not just a way of getting a computer to perform operations but rather it is a novel formal medium for expressing ideas about methodology. Thus programs must be written for people to read, and only incidentally for machines to execute.

I have used computational descriptions to communicate methodological ideas in teaching subjects in Electrical Circuits and in Signals and Systems. I am now working with Professor Jack Wisdom and others in developing and teaching a subject that uses computational techniques to communicate a deeper understanding of Classical Mechanics. Our class is targeted for advanced undergraduates and graduate students in Physics and Engineering. In our class computational algorithms are used to express the methods

[2] ibid.

used in the analysis of dynamical phenomena. Expressing the methods in a computer language forces them to be unambiguous and computationally effective. Students are expected to read our programs and to extend them and to write new ones. The task of formulating a method as a computer-executable program and debugging that program is a powerful exercise in the learning process. Also, once formalized procedurally, a mathematical idea becomes a tool that can be used directly to compute results.

We may think that teaching Engineering and Science is quite removed from daily culture, but this is wrong. Back in 1980 (a long time ago!) I was walking around an exhibit of primitive personal computers at a trade show. I passed a station where a small girl (perhaps 9 years old) was typing furiously at a computer. While I watched, she reached over to a man standing nearby and pulled on his sleeve and said: "Daddy! Daddy! This computer is very smart. Its BASIC knows about recursive definitions!" I am sure that her father had no idea what she was talking about. But notice: the idea of a recursive definition was only a mathematician's dream in the 1930s. It was advanced computer science in the 1950s and 1960s. By 1980 a little girl had a deep enough operational understanding of the idea to construct an effective test and to appreciate its significance.

At this moment in history we are only at the beginning of an intellectual revolution based on the assimilation of computational ideas. The previous revolutions took a long time for the consequences to actualize. It is hard to predict where this one will lead. I see one of the deepest consequences of computational thinking entering our society in the transformation of our view of ourselves. Previous revolutions have entered culture by affecting the way we think and the way we talk: We discuss economic phenomena in terms of "market forces." We talk about geopolitical developments as having "momentum." We think it is hard to accomplish an organizational change because of "inertia." In exactly the same way we will find computational metaphors sneaking into our vocabulary. We already hear ourselves describing some social interactions as "networking." We may "ping" a friend to see if he "acks," indicating that he is available. But these are still rather superficial. More telling is the fact that we can describe people and organizations as having "bugs," and that these can be "patched." Perhaps the most important consequence of computational thinking will be in the development of an understanding of ourselves as computational beings. Indeed, my personal experience as a computer programmer has made me aware that many of my own problems are bugs that can be analyzed and debugged, often with great effort, and sometimes patched.