EDUCATING COMPUTER SCIENTISTS: Linking the Social and the Technical

Batya Friedman
Peter H. Kahn, Jr.

Computer scientists need to be educated to understand some of the complex linkages between the social and the technical. In this sense, we offer an educational essay. But because educational practice is bounded by one's conception of the discipline, we first highlight the place and import of social concerns in computer science. We put forth the position that computer technologies are a medium for intensely social activity; and that system design—though technical as an enterprise—involves social activity, and shapes the social infrastructure of the larger society. Three educational approaches that link the social with the technical are discussed: standalone courses, practicums, and integrative methods. For each approach, we offer a variety of specific educational activities, showing not only what is possible, but what is currently being used successfully by research and teaching faculty. Finally, we suggest that as computing professionals we need to bring our humanitarian sensibilities further into our professional lives.

Imagine an individual who spends most evenings logged on to a computer network, engaged in written conversation with other like-minded individuals across the globe. Such activity may well bring to mind an intelligent person, technically proficient, globally aware, interested, and committed. While such a characterization may ring true now, a short time ago many people might have considered such a person a "computer nerd" who was uncomfortable with face-to-face conversation. What has changed? In part, more people are engaged in computer networking, so the activity as a whole seems less deviant from social norms. But more important, there has been a recognition that computer networking supports a valid, albeit unique, form of social interaction.

Where communication is concerned, most of us are willing to carve out a social niche in computer science. But we can and should go further. Technologies cannot be divorced from a social framework. Consider an analogy to architecture. Architects design structures that people use, including homes, schools, hospitals, businesses, and factories. By determining such physical environments, architects influence how people go about living their lives. Think about an architect planning to put a door between two offices in an office building. Even with something this simple, the architect faces structural considerations (e.g., will the wall stand once the door is in place?). Additionally, social considerations arise. Aesthetics is perhaps the most obvious (e.g., is the door's placement and design pleasing?). But other considerations are no less important. After all, the very question of such a door brings to the foreground issues of privacy vs. access. If both offices are meant to be relatively private spaces, perhaps no door is appropriate. If meant otherwise, then a door could support easy passage between the rooms, and easy communication between the inhabitants. A door with a one-way lock could prepare the way for a hierarchical relationship between, say, a receptionist and his or her boss. A door with a two-way lock requires mutual consent for access: a situation that arises sometimes between business partners. The width of the door also matters. Will the space accommodate people in wheelchairs?

What does this description of doors and architecture have to do with computing? As in architecture, much of computer science concerns the design and building of structures that shape the environment in which we live. Both disciplines also call for structural and social considerations. For example, the addition of a node to a computer network is not unlike the placement of a door in a building, since it too raises issues of privacy and access. While privacy is often attained through the familiar language of passwords and privileges, such restrictions can be at odds with claims to fair and equitable access. The National Information Infrastructure offers a pressing case in point: Who will have access to this federal "data highway," on what terms, and who decides? Will, for example, service be constrained by economic means? geography? education? At stake is nothing less than the social values that will be embedded within global telecommunications into the next century.

We are not the first to draw parallels between architecture and computer science. For example, in the 1980s Hooper [18] wrote substantively on the similarities between architecture and interface design. More recently, Adler and Winograd [1] introduced the idea of usability experts as akin to architects, "with one foot in the technical engineering domain and one in the human social domain." We would also extend the architecture analogy beyond interface and usability concerns to embrace fully those aspects of computer science that deal with artifacts for human use.

Just as a largely structural view of architecture would yield impoverished buildings so, too, a largely technical view of computer science leads to the impoverished design of computer systems. Our literature abounds with reports of computer systems that passed technical muster, but posed ethical concerns or made little sense for the social context of their use. We hear, for example, of police harassment of law-abiding U.S. citizens due to poor design and maintenance of information in the computer networks and databases of the National Crime Information Center [25]. In Sweden, a major airline (SAS) introduced an expert system (ES) in the maintenance process of their planes. Subsequently, repair quality declined—not because of technical shortcomings in the ES but because the system usurped understanding and responsibility from the airplane mechanics [5]. The tales go on, cutting across a diversity of fields, including business, communication, education, manufacturing, medicine, and the military.

Architects understand that their clients know something valuable about the human activities a specific building is intended to support. Thus, architects frequently engage their clients in the design process, minimally in the early phase of specifying what function the building is to serve and often at other key points in the design process. The same is possible in computer science, wherein we consult with the users on design considerations. This idea fits within a broader conception of participatory design (PD) [3, 14, 26] which, in its robust form, seeks not only to create better designs sensitive to the users' needs but to enfranchise and validate users in the design process. Through such participation, the idea of the computer scientist as keeper and definer of the technology gives way to a more embracing conception of the designer and user as not only complementing one another, but essentially linked to help create the environment within which we live and work.

Because computer system design involves social activity and shapes and impacts the larger society, we suggest that not only technical but social criteria be used in assessing the merits of a particular system and its use. Several areas have been increasingly discussed in the literature, namely safety, usability, and aesthetics. We propose one more that seems to us of particular import: Is the computer system designed to promote human responsibility for the consequences of computer-mediated action? (see [10, 20]).

Two overarching design considerations can promote such responsibility. First, we should design systems that do not diminish human agency. When, through computer use, humans are placed within largely mechanical roles, mentally or physically, human dignity is often eroded and individuals may consider themselves to be largely unaccountable for the consequences of their computer-mediated action. Second, we should design systems that do not masquerade as a human agent by projecting such attributes as intentions, desires, volition, consciousness, and free will. This is not to say that computers should not do some of the same tasks that humans do. But, to the extent that humans inappropriately attribute agency to computational systems, humans may well consider such systems, at least in part, to be morally responsible for the effects of computer-mediated actions.

Educational Practice that Links the Social and the Technical

If, as we claim, social and ethical concerns are an integral part of designing computer systems, then why are they frequently absent in computer science education? In some respects, their absence is not so surprising. After all, computer science emerged from departments with technical emphasis, including engineering and mathematics. However, in the last decade or so, under the rubric of human-computer interaction, there has been an increasing body of research investigating human considerations in computer system design. Much of this research addresses human computer use on the microlevel—on the level of an individual user in front of a single machine—and includes work on command languages, menu selection, direct manipulation, icon design, screen design, help systems, and input devices [17, 27]. Other research addresses computer use on the macrolevel, examining the effect of computer systems on human activity: for individuals, organizations, and society [7, 13, 14, 15, 28, 31]. In turn, colleges and universities have been called on to modify their computer science curricula. For example, in 1991 the ACM/IEEE-CS Curriculum Task Force for undergraduate computer science education formally made professional and ethics education one of its nine core strands [29].

Thus the charter is increasingly

clear: Link the social and the technical in computer science education. What is less clear is how to do so effectively.

Perhaps the most widespread approach has been to offer specific courses. Typically these courses follow standard lecture, seminar, or discussion formats with emphasis on computer ethics, legal issues, computing in the workplace, and other social analyses of computing (see [12] for a collection of course syllabi; see [4, 6, 8, 23, 24]). Such "standalone" courses guarantee time in the curriculum to address issues in depth, and can be taught by someone with expertise in the area. However, several shortcomings can be noted. As documented by Baum [2] in the context of the engineering curriculum, teaching about the social and ethical dimensions in a standalone course separates the social material from the technical, and can thereby convey an underlying message to students: social issues do not really count when you are doing your technical work. Moreover, for those students who think social issues should count in their technical work, standalone courses provide little actual guidance.

In response, some faculty have developed alternative approaches. One approach builds social and ethical issues into a practicum in which students encounter social and ethical concerns in the process of an actual design project, potentially for a real client. For example, in her teaching at Mills College, Friedman [9] had students work in small teams on pieces of on-going campus-based computer projects. One project, for example, involved library automation, another an interactive videodisc about mathematics for use in public spaces (e.g., shopping malls and science museums). Social and ethical issues emerged from students' own computing practice. In the videodisc project, for example, students came to recognize that their decisions about language usage and images had impact on who could or indeed would use their technology in a public space. Along similar lines, Hartfield and Winograd at Stanford developed a practicum on human-computer interaction. Small teams of students worked with industry mentors to analyze a work environment, then to design and implement a prototype user interface, and finally to evaluate the prototype with prospective clients [16].

Thus practicums can easily link social concerns and technical material. Such courses also help students develop the client-computer scientist relationship that sits at the core of PD practices and philosophy. If a practicum is not required, however, then not all students will elect it. Moreover, not all of the social issues we might want to discuss with students arise in a practicum.

Another, and certainly complementary, approach integrates social issues with traditional technical material in standard technical courses [22]. In our view, the integrative approach takes its most powerful form when social issues emerge from students' own technical work. Then connections can be made to similar issues in the larger computing community outside the classroom. A number of examples follow to illustrate what is possible in this area.

Consider teaching about counting loops, which is typical fare for introductory computer science courses. Gotterbarn (personal communication, August 24, 1993) at East Tennessee State University has taught this material by asking students to write the software for a cardiac pacemaker. The required specifications are as follows: If the heart stops beating for 10 seconds, the pacemaker's program should send a signal to give the heart one electrical shock. Medically, the idea is that this shock stops the heart from beating for an additional second, and then frees it to beat normally. In Gotterbarn's experience, many introductory students write a simple counting loop from 1 to 10, nested in an infinite loop. Unfortunately, these students neglect to take into account the duration of machine cycles, so their counting loops execute approximately every one-millionth of a second. Thus they programmed their pacemakers to provide shocks approximately every 10 one-millionths of a second. The effect on the wearer is self-evident: electrocution.

Gotterbarn uses such technical mistakes made by his students to discuss the difficulty of testing software in some real-life situations, and software engineers' responsibility for their technical work. Such mistakes also provide an ideal forum to study actual cases that raise comparable issues. For example, due to software problems, Patriot Missiles failed to intercept incoming Scud Missiles during the Persian Gulf War [30]. Here was one problem: In predicting the Scud's incoming flight path, the Patriot depended on a calculation that used its system's internal clock. Due to the way the Patriot's computer handled the intercept calculation, however, the longer the system ran continuously, the less precision in the calculation. Such imprecision contributed to, if not caused, failed intercepts. Testing failed to detect this error because the system's continuous run time during war conditions was underestimated.

In another assignment aimed at teaching the use of conditionals, Gotterbarn asked students to write software to control the vertical position of a 3,000-pound X-ray machine mounted on a post. Setting the position dial to 0 should move the X-ray machine onto a table at the bottom of the post, setting the dial to 15 should move the machine to the top of the post, and setting the dial to an integer between 1 and 14 should move the machine to that relative height. Students' programs usually work well, making reasonable use of CASE or nested IF-THEN statements. But then Gotterbarn asks his students how they would test such software, and relates this apparently true incident: An X-ray technician placed a patient on the table, walked out of the room, and then set the position dial to 0. The software and associated machinery lowered the X-ray machine to the table, crushing the patient. Here students' technical work leads to discussions about safety-critical systems, unanticipated consequences of systems in use, and accountability for software error. We might also add that such an assignment provides an opening to discuss the well-documented malfunctioning of Therac-25, in which software error and poor safety features resulted in the overradiation and death

of several cancer patients [21].

Technical activities can also provide the content for teaching about intellectual property rights. In his introductory computer science course Abelson at MIT (electronic communication, May 25, 1993), has students study and implement the RSA encryption technique. The students are then informed that they had encountered a patented method. In a class handout, he writes: "Although you could easily implement this [RSA] method as a program (and have done so), you are legally prohibited from developing a communication system that incorporates this method, and letting others use it, unless you obtain a license from that company that 'owns' the method."

Readings were issued to provide a larger context for intellectual property issues and for the RSA patent in particular, including information about an actual violation of the RSA patent. Students then considered an elaborated version of the following hypothetical scenario: Alyssa P. Hacker, an MIT undergraduate who took the introductory computer science course where she learned about the RSA method, worked in a local high school helping set up a computer system complete with networking capabilities. While still an undergraduate, she implemented the RSA method as part of the system and offered her system to people in the school to use in sharing files and messages, both within the school and with other schools. She also uploaded her programs to a national bulletin board. The result is that many people had access to Alyssa's implementation of the RSA method. Given the way in which files are redistributed, there is no way of knowing how many people or who exactly had access to Alyssa's file.

Eventually, the company that owns the patent to RSA learned of the school's distribution of the RSA method and wrote a letter claiming patent infringement. This is the first that Alyssa learned of the patent's existence. So, a meeting is arranged between Alyssa, a representative from the company holding the patent, and the school superintendent. After reading this scenario, students are challenged to role-play this meeting, changing roles at least once so they may see better the different perspectives on the issues. Although hypothetical, this scenario touches close to students' lives. With some force, it brings students into the complexity, ambiguity, and seriousness of the intellectual property issues that surround computer technology.

We now turn to teaching about bias in computer systems. Friedman and Nissenbaum [11] have identified three overarching categories of such bias: preexisting social bias, technical bias, and emergent social bias. Preexisting social bias has its roots in social institutions, practices, and attitudes. It occurs when computer systems embody biases that exist independently of, and usually prior to, the creation of the system. Technical bias occurs in the resolution of technical design problems that often arise due to limitations of the hardware, programming tools, or algorithms. Finally, social bias emerges in the context of the computer system's use, often when societal knowledge or cultural values change, or the system is used with a different population. These three categories of bias can be of use when teaching. Friedman, for example, in a course on data structures and algorithms asks her students at Colby College to design and implement a computer dating program. The technical material focuses on the use of records and linked lists. The issue of bias typically arises when students determine issues such as who will be included in the database and how individuals in the database will be searched. Some students assume only heterosexual users, for example, and their programs are sometimes critiqued by other students on the basis that the program results in the unfair exclusion of homosexuals, due either to oversight or to the programmer's preexisting bias against homosexuals. Other programs search for matches with a first-entered first-searched strategy, and thus unfairly favor those individuals who join the database earlier over those who join later. This instance of technical bias commonly arises because of students' choices of data structure and search strategy.

We have sketched three educational approaches that link the social and the technical: standalone courses, practicums, and integrative methods. We now call attention to an educational strategy that can enrich all three. Namely, have students study existing computer systems in actual use from a social perspective. At the University of California at Irvine, Kling and his colleagues do precisely this in a year-long sequence of courses on the social and organizational analyses of computing (Kling, electronic communication, September 10, 1993). In the final course of this sequence, teams of two or three students study organizational dimensions of computing in local organizations.

A comparable strategy is taken by Friedman, who has students choose an organization—a fast-food restaurant, a bank, a manufacturing plant, the school alumni office—most any will do, and research how computer technology was brought into the organization, supports the organization's goals, and shapes social interactions. Students can collect "data" through observations, surveys, and extended interviews. Their data can inform on such questions as: Who decided that computer technology would be good to have in the organization and for what reasons? Who decided on the type of technology? To what extent did workers and consumers participate in this process? What do managers and workers think about the technology's current use? Were there any unanticipated consequences from using the technology? Did any ethical issues arise, and, if so, what and how?

One group of students, for example, elected to study computer use in a small accounting firm. The firm used a computerized tax preparation program chosen for its ease of use and compatibility with existing hardware. In addition to these features (and unnoticed by the members of the firm at the time of purchase), the software kept a running tally of the number of tax forms completed by each accountant. This information on each accountant was revealed at the end of the tax season. Office discord followed. In response, the firm made a collective decision to "hide" this unsolicited information in subsequent years. Through reflecting on

Computer science education should not drive a wedge between the social and the technical, *but rather link both throughout the formal and informal curriculum.*

findings like these, students can develop greater sensitivity to the influence of computer system designs on social interactions, work, and organizations.

The preceding activities occur within the formal, structured educational curriculum. Yet many informal, unstructured activities support such a curriculum, and these deserve some attention. It is likely that computer science faculty already engage in some of these activities; and thus their implementation can build on current strengths. We sketch four broad areas for consideration (for further discussion by Friedman, see [9]):

a) Students directly encounter social aspects of computing through the policies that govern their own school computer use. Thus it can work well to involve students in establishing significant aspects of school computer policy. Areas for involvement could include allocation of computer time, promoting access to information, and establishing security for systems.

b) Faculty often use email to communicate with students about technical issues, and practical concerns, such as homework updates. But there is no need to stop there. Social issues can provide engaging content areas for discussion among class members. More broadly, it is possible to involve students in various bulletin boards, like the RISKS forum, that discuss the social aspects of computing. By such involvement, students informally recognize ways in which the social aspects of computing are embedded in the larger computing community.

c) Students often bring social and ethical issues related to computing to faculty members during office hours. How do faculty members respond? From our experience, students construct understandings about the field based partly on what we as faculty bring to the advising tables. Unduly shifting the discussion back to technical concerns can convey the message to students that social issues are of little import.

d) Departmental colloquia can at times include social topics. In our own experience at Colby College, colloquia on social aspects of computing have provided both students and faculty with common issues to discuss, in the classrooms, hallways, and beyond.

Conclusion

We have suggested that computer science education should not drive a wedge between the social and the technical, but rather link both throughout the formal and informal curriculum. We have also offered numerous activities, which can provide ideas for faculty as they shape their departmental studies. Equally important, the activities go some distance toward answering possible concerns. We have heard it said, for example, that "social aspects of computing are not important." They are. "Technically oriented students need not be versed in the social aspects of computing." They should. "Faculty in computer science do not currently teach social aspects of computing." Some do.

Finally, more akin to our own thinking, we have heard it said that computer science education needs to change if the field is to change. This is true. But let us not place too much emphasis on the academic educational experience, since, for education to change so must the field. For educational reform to take hold it is necessary that its goals, processes, and products be welcomed into the larger computing community. Perhaps less apparent to those outside of academic institutions, students, as they should, often look to computing in the world at large to help them determine what is "real" computer science. When they do, students need to see genuine ethical computing communities in practice—in our designs and dialogue, and in what garners prestige. Thus our discipline needs to readily embrace social issues: computing that supports basic freedoms of speech and privacy, computing that supports responsibility for the consequences of our technical work, computing that supports safety in the workplace, and access for the disenfranchised. The full list is not small. But to call for the social and ethical education of young computer scientists is a call to us all in our professional lives.

Acknowledgments

We thank Helen Nissenbaum and Doug Schuler for comments on an earlier draft of this article.

References

1. Adler, P.S. and Winograd, T.A. *Usability: Turning Technologies into Tools.* Oxford University Press, New York, 1992.
2. Baum, R.J. *Ethics and Engineering Curricula.* Hastings Center, Briarcliff Manor, N.Y. 1980.
3. Bodker, S. *Through the Interface: A Human Activity Approach to User Interface Design.* Erlbaum, Hillsdale, N.J., 1991.
4. Bynum, T.W., Maner, W. and Fodor, J.L., Eds. *Teaching Computer Ethics.* Research Center on Computing and Society, New Haven, Conn., 1992.
5. *Computers in Context.* [video]. California Newsreel, San Francisco, Calif., 1986.
6. Dunlop, C. and Kling, R., Eds. *Computerization and Controversy: Value Conflicts and Social Choices.* Academic Press, Inc., San Diego, Calif., 1991.
7. Ehn, P. *Work-Oriented Design of Com-*

puter Artifacts. Arbetslivscentrum, Stockholm, 1988.
8. Ermann, M.D., Williams, M.B. and Gutierrez, C., Eds. *Computers, Ethics, and Society.* Oxford University Press, Oxford, England, 1990.
9. Friedman, B. Courting culture in computer science. In *Teaching Computer Ethics.* Terry W. Bynum, Walter Maner and J.L. Fodor, Eds., Research Center on Computing and Society, New Haven, Conn., 1992.
10. Friedman, B. and Kahn, P.H., Jr. Human agency and responsible computing: Implications for computer system design. *J. Syst. Softw. 17* (1992), 7–14.
11. Friedman, B. and Nissenbaum, H. Discerning bias in computer systems. In *InterCHI '93 Adjunct Proceedings* (Apr. 24–29, Amsterdam). ACM/SIGCHI and IFIP TC.13, 1993, pp. 141–142.
12. Friedman, B. and Winograd, T., Eds. *Computing and Social Responsibility: A Collection of Course Syllabi.* Computer Professionals for Social Responsibility, Palo Alto, Calif., 1990.
13. Galegher, J., Kraut, R.E. and Egido, C., Eds. *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work.* Erlbaum, Hillsdale, N.J., 1990.
14. Greenbaum, J. and Kyng, M. *Design at Work: Cooperative Design of Computer Systems.* Erlbaum, Hillsdale, N.J., 1991.
15. Greif, I., Ed. *Computer-Supported Cooperative Work: A Book of Readings.* Morgan Kaufman, San Mateo, Calif., 1988.
16. Hartfield, B., Winograd, T. and Bennett, J. Learning HCI design—Mentoring project groups in a course on human-computer interaction. In *Proceedings of the SIGSCE '92 Technical Symposium of the ACM* (March 1992, Kansas City, Kans.). ACM/SIGSCE, 1992.
17. Helander, M., Ed. *Handbook of Human-Computer Interaction.* Elsevier Science Publishers, Amsterdam, 1988.
18. Hooper, K. Architectural design: An analogy. In *User Centered System Design,* Donald A. Norman and Stephen W. Draper, Eds., Erlbaum, Hillsdale, N.J., 1986.
19. Johnson, D.G. *Computer Ethics.* Prentice-Hall, Englewood Cliffs, N.J., 1985.
20. Kahn, P.H., Jr. and Friedman, B. Control and power in educational computing. In *Education/Technology/Power: Educational Computing as Social Practice,* Michael W. Apple and Hank Bromley, Eds. To be published.
21. Leveson, N.G. and Turner, C.S. An investigation of the Therac-25 accidents. *Computer 26,* 7 (1993), 18–41.
22. Miller, K. Integrating computer ethics into the computer science curriculum. *Comput. Sci. Ed. 1* (1988), 37–52.
23. Parker, D., Swope, S. and Baker, B.N. *Ethical Conflicts in Information and Computer Science, Technology, and Business.* QED Information Sciences, Inc., Wellesley, Mass., 1990.
24. Perrolle, J.A. *Computers and Social Change: Information, Property, and Power.* Wadsworth, Belmont, Calif., 1987.
25. Richards, E. Proposed FBI crime computer system raises questions on accuracy, privacy. In *Computerization and Controversy: Value Conflicts and Social Choices,* Charles Dunlop and Rob Kling, Eds., Academic Press, San Diego, Calif., 1991.
26. Schuler, D. and Namioka, A., Eds. *Participatory Design: Principles and Practices.* Erlbaum, Hillsdale, N.J., 1993.
27. Shneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, Reading, Mass., 1987.
28. Sproull, L. and Kiesler, S. *Connections: New Ways of Working in the Networked Organization.* The MIT Press, Cambridge, Mass., 1991.
29. Tucker, A.B. Computing curricula 1991. *Commun. ACM 34,* 6 (1991), 69–84.
30. U.S. General Accounting Office. *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia* (GAO/IMTEC-92-26). Washington, D.C., 1992.
31. Winograd, T. and Flores, F. *Understanding Computers and Cognition: A New Foundation for Design.* Addison-Wesley, Reading, Mass., 1986.

CR Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education, curriculum, information systems education, self-assessment*; K.4.2. [**Computing Milieux**]: Computers and Society—*Social issues*; K.7.m [**The Computing Profession**]: Miscellaneous—*Codes of good practice, ethics*

General Terms: Human Factors

Additional Key Words and Phrases: Computer science education, computer ethics, human-computer interaction, information system design, social implications of computing, undergraduate computer science education

About the Authors:

BATYA FRIEDMAN is Clare Boothe Luce Assistant Professor of Computer Science in the department of Mathematics and Computer Science at Colby College. Current research interests include information systems design, human-computer interaction, and technology and ethics.

PETER H. KAHN, JR. is an assistant professor in the department of Education and Human Development at Colby College. Current research interests include moral development, technology and ethics, and the human relationship to nature.

Authors' Present Address: Colby College, Waterville, ME; 04901; email: (beginning September 1994): {b_friedm, phkahn}@colby.edu