

Computing Education and the Information Technology Workforce

Eric Roberts

(with the endorsement of the ACM Education Board)

March 1, 2000

The Committee on Workforce Needs in Information Technology was formed in response to “concern that lack of an adequate, well-trained workforce may inhibit the capacity of the United States to remain competitive” in information technology (IT) [25]. The purpose of this white paper is to examine the labor shortage in IT, drawing on our experience not only as experts in computing, but also as educators in the computing field.

As we argue in the rest of this paper, we believe that the following statements are true of the labor economy in IT today:

- Critical shortages exist in particular specialty areas within the IT profession, specifically those directly associated with software development.
- The productivity of software developers—which we use as a generic designation for programmers, software engineers, systems analysts, and other professionals involved in the creation of software—varies enormously among individuals.
- Education is an important force in the intellectual development of most highly effective software developers.
- The educational system that produces many of the high-quality software developers is itself threatened by the enormous demand from industry for top talent.

1. Background

As noted in the summary of the project on Workforce Needs in Information Technology [25], there is widespread concern in the computing industry that the supply of well-trained computing professionals is significantly less than the current demand. Employers report critical labor shortages in certain IT areas, primarily those that are directly involved in software development. A 1997 report from the Information Technology Association of America found that 190,000 jobs in the IT industry remained unfilled because of a shortage of qualified workers [18]. After interviewing 441 CEOs of the fastest growing U.S. companies, Coopers & Lybrand reported that 66 percent viewed the shortage of qualified workers to be their greatest concern [8]. Other studies—including a detailed analysis issued by the Computing Research Association (CRA) [15] and two reports from the Office of Technology policy in the U.S. Department of Commerce [27, 28]—provide additional evidence to support the existence of an IT worker shortage.

Despite such evidence, the very existence of an IT labor shortage remains controversial. The authors of the 1999 CRA report were unable to agree on whether the situation facing the IT industry represents a significant labor shortage or a less serious, more ephemeral tightness in the labor market. The General Accounting Office (GAO) reviewed the methodology of the 1998 Department of Commerce report and identified “serious analytical weaknesses that undermine the credibility of the report’s conclusion” [32]. Critics within organized labor and some academics have expressed skepticism about the reality of such a crisis, arguing that any worker shortage in IT has been greatly exaggerated. In his congressional testimony against increasing the number of visas under

the H-1B program, Professor Norman Matloff of the University of California at Davis argues that the IT labor shortage is simply a myth perpetuated by an industry intent on keeping labor costs low [21]. Matloff observes that many of the companies that claim to face a labor shortage hire a very small percentage of their applicant pool, suggesting that the shortage cannot be as severe as those companies claim.

Why is there such profound disagreement on this issue? In part, the problem comes from looking at the IT labor market in an oversimplified way, without considering the specific character of work in the IT profession. We believe that a complete analysis of the labor market in IT requires not only an understanding of conventional analytical techniques from sociology and labor economics, but also a detailed sense of what work in the IT profession involves.

The importance of considering the nature of work in the IT profession is underscored by an interesting observation in the 1999 CRA report [15]. In Chapter 4, the authors of the CRA study note that “where (most) IT professionals see a shortage, (most) social scientists see a tight labor market.” It is certainly possible that IT professionals—who often lack a detailed understanding of the underlying economics—might fail to characterize the problem in precise sociological terms. It is, however, equally possible that social scientists—who often lack a detailed understanding of work in the field—might blur important distinctions to such an extent that they are unable to understand the specific nature of the IT workforce.

We believe that the failure to reach agreement on the existence of an IT labor shortage comes from the following overgeneralizations in traditional analysis:

- *Looking at the IT profession as a whole makes it difficult to understand the dynamics of particular specialty areas in which critical shortages exist.* The statistics collected by the Department of Labor about jobs in the IT profession combine many different job classifications into a single category. For many of these job classifications, there is an adequate supply of labor. Changing labor needs in these areas, moreover, are easily accommodated by standard economic forces in the labor market. In other segments of the IT profession, significant shortages exist. For the most part, the specialty areas that are in shortest supply are those that contribute directly to the creation of software. These specialties include such job titles as software engineer, software designer, systems analyst, and programmer. Because these areas give rise to the greatest concerns over labor supply, we focus our attention in the rest of this paper on the problems facing *software development* in preference to the more encompassing, but therefore less revealing, category of *information technology*. Although other IT job categories—such as network administrators, system administrators, and web designers—may also be in short supply, we have relatively little concrete data about those specialties and have therefore chosen to concentrate on the software development area.
- *Individual workers in the software-development area are by no means interchangeable.* Most economic analyses of the labor market in an industry depend largely on counts of workers with particular specialties, assuming that individual workers with the necessary skill sets and experience are essentially equivalent. This argument—a legacy of the shop-floor management theories of Frederick Taylor—does not apply to most skilled workers, because each worker acquires specialized expertise that cannot easily be transferred. In most software-related job categories, where individuals workers may vary in productivity by orders of magnitude, the assumption that workers are interchangeable is even less justified and leads to entirely insupportable conclusions.

- *Strategies adopted by the industry to identify and hire the talent they need are motivated much more strongly by the need to attract highly productive individuals than by a desire to reduce labor costs.* Many economic analyses of the computing industry look at corporate hiring policies as if the key goal is to reduce labor costs. In many cases—such as the widely reported tendency of firms to hire younger workers as opposed to older, more experienced employees seeking to reenter the job market after a layoff or other dislocation—any analysis based on the minimization of labor costs provides little insight into the problem and does not typically explain the actual behaviors observed. The industry strategy can only be understood in light of the specific characteristics of the software labor market, most notably the high variance of individual productivity.

2. The impact of individual differences in productivity

From early in the history of modern computing, studies of the computing workforce have revealed remarkably high variability in the productivity of individual software developers. The landmark study in this area was published in 1968 by Sackman, Erikson, and Grant [31]. In their study, programmers with the same level of experience exhibited variations of more than 20 to 1 in the time required to solve particular programming problems. Beyond this high variability in coding time, the study revealed that individual programmers showed highly correlated differences in other metrics that contribute to overall productivity, in the sense that the best programmers were not only able to complete the problem in less time, but in so doing typically produced programs that had fewer errors and were more efficient in both running time and utilization of memory. Thus, the best programmers were in fact *more* than 20 times as effective. More recent studies [4, 10, 11] have reaffirmed the results of the Sackman study and conclude that the best software developers are relatively few in number, but much, much better than the average.

If anything, the variability in productivity is actually greater today than it was in the early years of computing when the Sackman study was issued. Although it is difficult to determine precisely what skills and temperaments correlate with high individual productivity, one important characteristic that many highly productive software developers share is an ability to keep the details of a large, complex system in their heads. With the increased emphasis on interactive user interfaces and market demand for more sophisticated features, modern systems have become much larger and more complex than their counterparts of a generation ago, which tends to increase the gap in productivity. Some senior managers at major software companies claim that the variability in individual productivity may indeed have reached the 100 to 1 level. But no matter whether the variation in productivity is 100 to 1 or the more conservatively estimated 10 to 1, there is widespread agreement throughout the industry that the range is large.

While the last few paragraphs have focused on the most talented software developers, productivity variations can be just as important at the other end of the spectrum. The Sackman study found that “one poor programmer can consume as much time or cost as much as 5, 10, or 20 good ones.” In some cases, software developers who fall at the low end of the productivity curve may be essentially nonproductive or even counterproductive in terms of their contribution to a software development project. In a 1998 essay on “problem programmers,” software engineering expert Steve McConnell concludes with the following warning:

Problem programmers are often viewed as having “low productivity,” but both software research and software experience suggest that such an assessment is too optimistic. Next time you need to improve productivity, don’t look for what you can add, look for what you can take away. [22]

The variation in productivity is the critical dynamic that underlies employment policies in the software industry. Companies are under intense competitive pressure to identify and hire those extraordinarily talented individuals at the high end of the productivity curve. Particularly for startups employing a small team of implementers, hiring just one extremely productive person can make the difference between success and failure. After all, if the best software developer can do the work of 10, 20, or even 100 run-of-the-mill employees, a single-person company that attracts such a superstar can compete effectively against a much larger enterprise. Thus, companies whose business depends on software production will try to hire applicants from pools in which the likelihood of finding the most talented individuals is high, such as graduates from top computer science departments, successful participants in collegiate programming contests sponsored by the ACM and similar organizations, or entrepreneurs who have developed successful freeware and shareware systems on their own. Competition to attract employees from these populations is intense. Companies likewise have a strong incentive to avoid problem programmers and are unlikely to hire applicants whom they fear are at the low end of the scale.

If you think about hiring policies in light of the high variability in productivity, the prevailing practices of the industry make considerable sense. As Norman Matloff observes, software companies hire only a small percentage of their applicant pool, while continuing to complain of a shortage of talent. Given that only a modest percentage are likely to fall into the highly productive range and that some may turn out to be counterproductive, such selectivity seems quite rational. The alternative explanation offered by Matloff—that companies seek to reduce labor costs by hiring newly minted college graduates rather than more expensive older workers—just doesn't square with the observed corporate behavior. Newly minted college graduates from the top computer science programs are offered enormous salaries because of the intense competition for those students. The older workers who are interested in those same positions would be willing to work for considerably less than what the entry-level recent graduates receive. Reducing labor costs is not the central issue.

So why might companies be reticent to hire more senior workers with years of experience in the field? One does hear stories about software developers in their 30s—who may have been downsized out of their previous job or who may feel dissatisfied with their prospects for advancement in their current position—being unable to find good employment opportunities, even in an industry that claims to be desperate for talent. How can this happen? The fundamental reason is that such candidates are relatively unlikely to be the ones who occupy the high end of the productivity scale. Employees who have demonstrated their ability to work at ten times the average rate of productivity are unlikely to be downsized out of their positions or frustrated by limited advancement opportunities. Since such individuals are enormously valuable, companies will go to great lengths to keep them employed and happy. People who have worked in the industry for ten years and then find themselves in need of a job are not likely to be in that class.

The preceding paragraph should not be read as justification for corporate hiring policies that discriminate against older workers. The current employment policies of high-tech firms do create economic dislocations and exact a social cost that public policy needs to address. The point of the illustration is simply to underscore the importance of taking into account variability in productivity when analyzing the dynamics of the labor market. Traditional analyses of the IT workforce typically ignore the effects of individual differences and act as if workers in this profession are largely interchangeable, in the way that bricklayers or assembly-line workers are likely to be. The image of these job categories creates the wrong metaphor. It may be more reasonable to view software development as similar to artistic endeavors. One would not, for example, use statistics

about the number of fiddle players in a population to generalize about problems that a symphony director might have hiring a top-notch concertmaster. In his column for the *Los Angeles Times* [5], Gary Chapman observes that “the software profession is beginning to resemble professional sports or the movie business much more than engineering. Exceptionally gifted programmers are referred to as ‘talent,’ like movie stars, and some have incomes that make those of film stars look puny.” Wide variations in productivity are an essential characteristic of the software industry, and it is impossible to understand the economics of the associated labor market without recognizing the importance of this factor.

3. The role of computing education

So where do highly productive software developers come from? Is this kind of talent innate or formed at an early age? Or is it something that one acquires through education and training? As with most debates over the primacy of nature versus nurture, this question is difficult to resolve entirely in one direction or the other. The fact that college graduates with almost identical backgrounds can vary widely in their productivity indicates that individual qualities—which might be native intelligence, basic temperament, or simply a will to succeed—must play some sort of role. Conversely, some extremely talented software developers have achieved enormous success without completing their formal education. Classic examples include Steve Jobs and Bill Gates, who dropped out of Reed and Harvard, respectively. Thus, formal education in computing is neither a necessary nor a sufficient predictor of productivity in the software development industry.

At the same time, there is considerable evidence to suggest that computing education does play an important role in the development of highly productive software developers, in spite of the existence of well-publicized exceptions. Most people who work in the software industry are college graduates. While a majority did not major in computer science, most have taken some computing courses as undergraduates, regardless of their formal major. Thus, most workers in the software industry have had some exposure to computing education as undergraduates.

The field of software development has evolved in recent years in ways that make education even more important than in the past. In the early days of the field, a developer could go a long way inventing an entire system from scratch. Today, in this era of more complex systems and greater reliance on existing bodies of code, the body of knowledge that a software developer must master has grown considerably. Moreover, successful software projects now require much greater attention to issues of project management and software engineering discipline that individuals are unlikely to discover on their own.

As teachers in universities and colleges, we appreciate the value of a computing education through our students’ experiences. There are some students who enter college already endowed with enormously well-developed skills and high individual productivity. There are others who never seem to reach this level of skill no matter what steps we take as educators. These cases, however, are the exceptions. Most students arrive with weak software development skills and improve dramatically over the course of their undergraduate program. Some of these students—including those who had no intention of majoring in computer science until they discovered by taking introductory courses that they had an aptitude for the subject—become the highly productive individuals that industry is so eager to recruit. Without the education they received as undergraduates, many of these students would never have attained this level of expertise, and the pool of highly talented software developers would be reduced accordingly.

Perhaps the best evidence for the proposition that college-level education in computer science and related fields does produce people with the requisite skills and talent is the

fact that software companies recruit most intensively at computer science departments in top colleges and universities. That companies recruit heavily in such institutions and that the competition for new graduates from those departments is quite stiff is not particularly controversial. Indeed, Norman Matloff cites the fact that firms focus their recruiting on a relatively small number of top schools as evidence that no real shortage exists. As we argue in section 2, companies—which are extremely anxious to hire the most productive software developers—focus their recruiting efforts on those populations that have the greatest incidence of such top performers. The fact that companies recruit most heavily at highly rated universities and colleges indicates that the industry places considerable value on the education that students in those institutions receive.

4. The challenge for academic institutions

The fact that their graduates are in high demand does not mean that academic programs in computing are doing everything right. Academic departments in computer science and related disciplines are often criticized for a variety of perceived weaknesses, such as the following:

- Devoting too much attention to theoretical topics with little practical application
- Allowing curricula to become out of date with respect to technological advancements in the field
- Providing students with far too little experience in the practical techniques of building large systems
- Offering poorly designed introductory courses that do not attract good students into the discipline
- Failing to place sufficient emphasis on the nontechnical abilities that students need to work effectively in the field, including communication skills, management strategies, and the dynamics of working in a group.

Most educational institutions could do more in each of these areas.

At the same time, educational institutions face significant challenges that make it difficult to address these problems as effectively as one would like. Some problems arise from the institutional culture of colleges and universities. Because faculty tend to occupy their positions for a relatively long period of time and turnover is therefore modest, educational institutions are slow to change—a characteristic that is problematic for rapidly evolving fields like computer science. When new areas rise in prominence, it is often difficult for academic institutions to shift the necessary resources quickly enough to get a jump on the new area. Other problems, however, are difficult for academic institutions to resolve on their own. For example, the enormous rate of change in computing forces faculty to integrate new material into the curriculum much more quickly than in other disciplines.

The biggest problem facing computing programs in colleges and universities today, however, is a shortage of qualified faculty. The problem is in many ways the same as that facing the industry, in that both are competing for the same relatively small pool of highly productive individuals. Feeling incapable of attracting enough top talent to meet existing needs, industry is seeking to hire extremely competent people wherever they happen to exist, including those who are actively engaged in the educational process. Many companies seek to hire university faculty away from their institutions and recruit graduate students before they complete their degrees. As a result, there has been significant shrinking of the pool of available faculty just as enrollments in computing disciplines have begun to skyrocket. This phenomenon—which was first described in the early 1980s—has become known as the *seed-corn problem* [12]. In the previously cited

report from the Computing Research Association [15], the current incarnation of the problem is described as follows:

Many educators, industrial laboratory leaders, and government science officials are concerned that the high industrial demand for information technology (IT) workers will siphon out of the educational systems many students who would otherwise pursue an advanced degree. This diminishes [the] pool of people who will join the university faculties that perform basic research and teach the next generation of students. This problem is compounded when industry also successfully recruits current faculty members, including junior faculty who would become the academic leaders of the profession in the coming decades. This is known as the “seed-corn” problem—an analogy to those who consume too much of this year’s crop, reserving too little for next year’s planting.

The attrition of faculty and graduate students away from academic computer science has begun to attract attention in the popular press [2, 5, 17]. The severity of the problem is underscored by the following excerpt from a *Chronicle of Higher Education* article entitled “Computer scientists flee academe for industry’s greener pastures” [33]:

High-paying, fast-paced jobs in the computer industry are attracting both seasoned academics and newly minted Ph.D.s who, in the past, would have opted for careers in higher education. The upshot: Computer-science and computer-engineering departments are suffering a serious shortage of professors at a time when undergraduate enrollments are booming.

Many departments are losing professors faster than they can hire them. The University of Illinois at Urbana-Champaign recruited five new professors in electrical and computer engineering to start this fall, but lost five others who were already on its faculty. The University of Washington recruited four scholars to its department of computer science and engineering but lost five. Cornell hired three but lost six.

For computer science departments, the faculty shortage problem has reached crisis proportions. Drawing on data from a survey they conducted, Paul Myers and Henry Walker report that “only about half of the open tenure-track positions were filled” in 1997-98 [24]. For the current 1999-2000 academic year, the supply of faculty candidates has declined further, to the point that there will be candidates for only a third of the advertised positions. This situation is completely turned around from the typical pattern for academic searches in other fields, in which hundreds of applicants may apply for a single position.

Experience from the similar crisis of the 1980s shows how dangerous such faculty shortages can be when they occur in economically critical areas such as computer science. In 1983, Kent Curtis of the National Science Foundation issued a comprehensive report on the problems facing academic computer science [9]. In the following passage, Curtis argues that the term *crisis* was indeed justified by the extent of the problems:

We must conclude that the educational institutions of the country cannot obtain the labor they need and have poor prospects of finding it in the near future. They face a real crisis. The migration of student interest is working against them, not in their favor, and the job mobility which allows so many people to enter computer professions in business, industry and government is not effective for educational institutions because of their highly specialized job requirements.

Curtis based his evidence for the existence of a crisis on the following observations:

1. Students are not entering graduate school but are being lured by attractive salaries and professional opportunities at the bachelor’s level.

2. Graduate students are leaving graduate school without completing their Ph.D.s.
3. Faculty are leaving academia for industry.

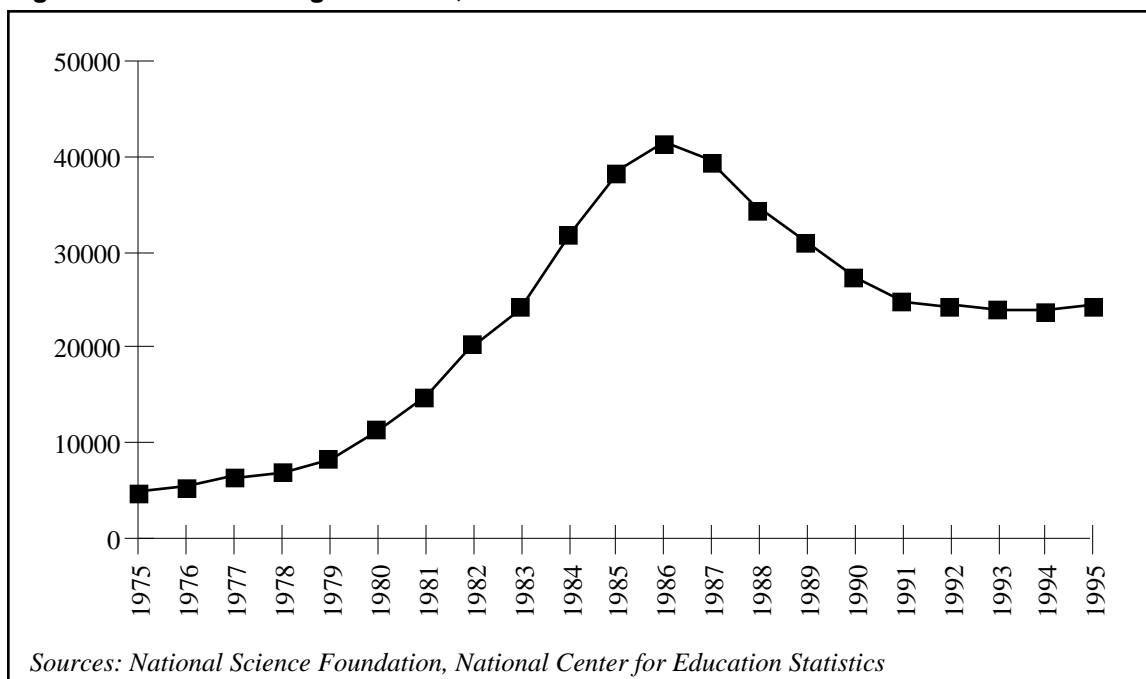
Academic institutions suffer from each of these problems again today.

In the 1980s, academic institutions responded to the problem of growing student demand in the face of faculty shortages by attempting to meet that demand with their existing faculty. As Curtis reports, this strategy often had negative long-term effects:

[An NSF study on the faculty hiring crisis reveals that] 80% of the universities are responding by increasing teaching loads, 50% by decreasing course offerings and concentrating their available faculty on larger but fewer courses, and 66% are using more graduate-student teaching assistants or part-time faculty. . . . In brief, they are using a combination of rational management measures to adjust as well as they can to the severe manpower constraints under which they must operate. These measures make the universities' environments less attractive for employment and are exactly counterproductive to their need to maintain and expand their labor supply. They are also counterproductive to producing more new faculty since the image graduate students get of academic careers is one of harassment, frustration, and too few rewards.

The faculty shortages of the early 1980s may well have precipitated the dramatic collapse in the number of undergraduates majoring in computer science. After rising quickly for nearly a decade, bachelor's degrees in computer science declined sharply after 1986, as illustrated by the chart in Figure 1. Although there are many factors that may have contributed to the dramatic drop in the number of computer science majors—which occurred several years *before* the economic downturn in the computing industry—some have argued that the principal reason was not that students *chose* other fields but that they were actively *prohibited* from majoring in computer science by departments that lacked sufficient resources to accommodate them [30]. These restrictions led to a decline in the number of college graduates with computing degrees, which in turn reduced the pool of talent available to industry. Given the dependence of the modern economy on software

Figure 1. Bachelor's degrees in CS, 1975-1995



technology, it would be disastrous if academic computer science departments endured a similar collapse today.

5. Strategies to address the current crisis

It is usually easier to analyze a problem than it is to solve it. The shortages in specific sectors of the IT workforce and the faculty hiring crisis are both serious problems that may prove difficult to solve. There are, however, some affirmative steps that academia, industry, and government can take to mitigate the effects of the faculty shortage and the eventual implications of that shortage on the workforce as a whole.

To reduce the problems associated with the faculty shortage, colleges and universities can adopt the following strategies:

- *Make sure that faculty in computing disciplines receive the support they need to stay in academia.* The studies undertaken by the NSF in the 1980s found that most faculty who left academia for industry did not cite economic motivation as the primary reason for the shift [9]. Instead, those faculty identified a range of concerns about the academic work environment—huge class sizes, heavy teaching loads, inadequate research support, the uncertainty of tenure, and bureaucratic hassles—that the NSF study refers to collectively as “institutional disincentives.” As enrollments in computing courses rose, these institutional disincentives increased in severity, to the point at which faculty became overwhelmed and sought other opportunities elsewhere. Institutions must learn from the experience of that period and take steps to protect the time and resources available to faculty in these areas.
- *Integrate information technology throughout much more of the curriculum.* In many institutions, it will be impractical to meet student demand for IT education entirely within computer science departments that have traditionally represented the core of the discipline. Today, computing and information technology have relevance to most academic disciplines, and it is important to improve the level of IT education throughout the entire curriculum, as recommended by the 1999 *Being Fluent with Information Technology* report issued by the Computer Science and Telecommunications Board [7].
- *Create academic positions that focus on teaching.* Although the current hiring crisis has led to a certain relaxing of traditional norms, most faculty positions still require a Ph.D. and involve both research and teaching. Given the large applicant pools in other disciplines, it is usually easy to find scholars who are skilled in both teaching and research. In computer science, where there are more faculties than candidates, institutions cannot afford to be so selective. It is not necessary for every institution to maintain a research program in computer science. At the same time, it is hard to imagine that any college or university today could get away without offering courses in this area. Opening faculty positions to those who enjoy teaching but are not interested in academic research increases the size of the available pool.
- *Get undergraduates involved in teaching.* The crisis in computer science education arises from the fact that there are too few teachers to serve the needs of too many undergraduates. One of the best ways to meet the rising student demand is to get those undergraduates involved in the teaching process. Using undergraduates as teaching assistants not only helps alleviate the teaching shortfall but also provides a valuable educational experience to the student assistants. Many schools—including Brown, Georgia Tech, Harvard, Stanford, the University of Arizona, and the University of Virginia—have major programs to recruit and train undergraduates as teaching assistants [29].

Industry also has an important role to play in addressing the IT workforce problem. Most importantly, industry needs to support—even more that it already does—the educational institutions that produce most of the employees on whom that industry depends. There are many ways in which individual companies can support education without jeopardizing their economic viability:

- *Provide financial support for educational institutions, departments, and students.* The computing industry has generated enormous economic value in recent years. In today's high-tech economy, money is rarely the critical resource; talent is. Investing money to develop talent is certainly an option that helps both the educational system and the companies themselves.
- *Encourage software developers to take adjunct teaching positions in colleges and universities.* Although companies are understandably reticent to lose the services of talented developers, it is sometimes possible to offer as a employment benefit the opportunity to engage in occasional teaching. In most cases, making this option available to employees results in a win-win dynamic. Employees have a chance to take a break from their usual routine at the same time that they bring practical experience into the classroom.
- *Expand internship and cooperative programs so that students, in effect, acquire part of their education in an industrial setting.* Although universities and colleges have many strengths, there are important aspects of a computing education that are easier to provide in an industrial setting. It is, for example, probably easier to learn practical software engineering methodologies during an internship than in a conventional academic setting. With effective cooperation, industry could assume responsibility for some aspects of the educational process, leaving academic institutions free to focus on those aspects of a computing education for which they are most appropriate.

Although academia and industry can cooperate in many ways to the benefit of both, it is important to recognize that the interests of the two communities may sometimes appear to be in conflict. At one level, the easiest solution to the seed-corn problem would be to convince industry not to hire faculty away from universities or lure students away from academic jobs. It is, after all, in the long-term interest of the industry to keep the educational system strong to ensure that the supply of talented, well trained developers will continue into the future. By cooperating with academia toward this common purpose, both will come out better in the end.

The problem with this optimistic scenario is that it depends on the collective willingness of a large community to put long-term public good ahead of short-term individual gain. Given the current shortage and the extraordinarily high value of the most productive workers, individual companies have an incentive to hire as many talented software developers as they can, even though that action has the long-term effect of diminishing the supply of such talent [14]. This problem is a classic example of what Garrett Hardin identified as the *tragedy of the commons* in 1968 [16]. Hardin explains the problem by asking his readers to picture a sheep-grazing pasture open to all. What happens when that pasture, or commons, reaches its capacity? Each individual shepherd has an incentive to introduce more sheep into the commons, even though it cannot support the increase. After all, the profit from the extra sheep remains with the individual shepherd, while the cost of overgrazing the pasture is, in the long run, shared by all. In game theory, this problem is often known as a *prisoner's dilemma*, in which the optimal choice for each individual is not the best choice for the community as a whole.

To get a sense of how these concepts apply to the computing industry, put yourself in the position of the CEO of a new startup with lots of venture capital and enough good ideas

to make a go of it. The critical problem for your company is attracting the software development talent necessary to turn those ideas into reality. Suppose that you find some student who seems to have the necessary skills and temperament to be just the highly productive developer you need, but who currently wants to pursue an academic position instead. Even if you understand the importance of having such students go on to academic careers, can you afford not to recruit that student for your own company? In a world where the majority of startups fail but some succeed remarkably well, your entire success may depend on making successful hires whenever you can. Your dilemma is exacerbated by the fact that the most negative outcome is not that such a student might enter academia—which, after all, has long-term benefits in which your company, should it survive, will share—but that the student will be tempted by an offer from one of your competitors. Since the long-term gain is realized only if other players agree to cooperate, the situation you face is a classic prisoner's dilemma. In such a situation, you may feel forced to hire anyone who is likely to be an extraordinarily productive developer, even though you recognize that such a decision may have long-term negative consequences for the industry as a whole.

When a community faces a tragedy-of-the-commons problem, the best strategy is for that community—often through some public entity that is at least nominally independent of the market forces that create the problem in the first place—to impose constraints that restrict the ability of individual participants to harm the community as a whole. In the case of the seed-corn problem, both federal and state governments have a role to play in ensuring that market forces serve the long-term public good.

In particular, government can take the following actions:

- *Create incentives to make teaching positions more attractive.* Both federal and state governments can help increase the attractiveness of faculty positions in the IT field. Financial support to make salaries more competitive with those in industry is one approach, but there are other approaches as well. Increasing the level of research support for junior faculty will help them establish their research programs and reduce the problems they face in their early years, when the temptation to jump to industry may be strongest. It may also be possible for governmental agencies to offer additional support for teaching assistants in undergraduate computing courses, which tend to be extremely large.
- *Provide increased support to graduate students in IT.* The most disturbing factor associated with the faculty shortage in IT areas is that the number of students entering Ph.D. programs in these areas is dropping, which suggests that the future supply of faculty may be even smaller. Additional support for graduate students would reduce the gap between the living standard of a doctoral student and that of a recent graduate working for the industry, which might encourage more students to pursue academic careers.
- *Offer incentives to academia and industry to implement reforms in their own structure that support IT education.* In the earlier parts of this section, we outline a set of actions for academia and industry that would reduce the severity of the current workforce shortages. Through tax policies or other incentive programs, governments could encourage both academia and industry to adopt these measures. For example, tax breaks could be established for companies that allow their employees to serve as adjunct faculty or that develop programs for educational internships.
- *Make the cost of depleting future resources more explicit.* As noted earlier in this section, the seed-corn problem fits the classic paradigm of the tragedy of the commons. Companies make individual decisions to attract as much of the limited pool of top talent as they can because the long-term costs of those decisions are not immediately

visible. Moreover, the costs of this sort of “overgrazing” are shared among the entire industry, while the benefits accrue only to the individual company that makes the hire. As such, the long-term cost becomes what economists call an *externality* in the context of the hiring process. One approach to reducing the negative consequences of such externalities is to expose their cost. Although the details are certainly complicated, it might be possible to impose a tax on companies whose hiring strategies reduce the capacity of the educational infrastructure. In this way, the cost associated with actions that work against the good of the whole would be borne by those who derived the associated benefit.

6. Conclusions

In this paper, we have presented evidence to show that the IT workforce shortage is real, at least for those areas directly involved in the creation of software. Several studies throughout the history of the field have shown that individual productivity varies widely among software developers, possibly by as much as two orders of magnitude. Given the extraordinary value of people at the high end of this productivity scale, employers compete strongly for individuals who have those skills. Although there are a few well-publicized exceptions, the vast majority of the highly productive software developers have attained and refined their skills through education.

Unfortunately, the same competitive pressures that generate intense competition for workers tempt faculty and graduate students to abandon academia for more lucrative positions in industry, leading to a severe faculty shortage. In this paper, we have proposed several actions that academia, industry, and government can take to alleviate the current crisis. Sustained growth in our economy depends on finding a solution to the IT workforce problem. To this end, industry, academia, and government must work together to develop strategies that advance the good of society as a whole.

References

1. Clifford Adelman. Leading, concurrent, or lagging? The knowledge content of computer science in higher education and the labor market. Washington, DC: U.S. Department of Education, 1997.
2. Ethan Bronner. Voracious computers are siphoning talent from academia. *The New York Times*, June 25, 1998, page A1.
3. Frederick P. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering* (anniversary edition). Reading, MA: Addison-Wesley, 1995.
4. G. Edward Bryan. Not all programmers are created equal. In Richard Thayer, *Software Engineering Project Management* (second edition), IEEE Computer Society, 1997.
5. Gary Chapman. Gold rush mindset undermining programming field. *Los Angeles Times*, February 14, 2000. <http://www.latimes.com/business/columns/dnation/>.
6. Computer Science and Telecommunications Board. *Realizing the Information Future: The Internet and Beyond*. Washington, DC: National Academy Press, 1994.
7. Computer Science and Telecommunications Board. *Being Fluent with Information Technology*. Washington, DC: National Academy Press, 1999.
8. Coopers & Lybrand. High-tech growth firms scale back 12-month growth outlook to match current pace: Worker shortage stifles growth. *Trendsetter Barometer*. January 7, 1998. <http://www.colybrand.com/eas/trendset/170.html>.

9. Kent Curtis. Computer manpower: Is there a crisis? Washington, DC: National Science Foundation, 1983. <http://www.acm.org/sigcse/papers/curtis83/>.
10. William Curtis. Substantiating programmer variability. *Proceedings of the IEEE*, July 1981.
11. Tom DeMarco and Timothy Lister. Programmer performance and the effects of the workplace. *Proceedings of the 8th International Conference on Software Engineering*. IEEE Computing Society Press, August 1985.
12. Peter J. Denning. Eating our seed corn. *Communications of the ACM*, June 1981.
13. Peter J. Denning, Edward Feigenbaum, Paul Gilmore, Anthony Hearn, Robert Ritchie, Joseph Traub. A discipline in crisis. *Communications of the ACM*, June 1981.
14. Peter J. Denning. Our seed corn is growing in the commons. *Information Impacts Magazine*, March 1999. http://www.cisp.org/imp/march_99/denning/03_99denning.htm.
15. Peter Freeman and William Aspray. The supply of information technology workers in the United States. Washington, DC: Computing Research Association, July 1999. http://www.cra.org/reports/wits/it_worker_shortage_book.pdf.
16. Garrett Hardin. The tragedy of the commons. *Science Magazine*, December 13, 1968.
17. Amy Harmon. With boom in high technology, software jobs go begging. *The New York Times*, January 13, 1998, page A1.
18. Information Technology Association of America. *Help Wanted: The IT Workforce Gap at the Dawn of a New Century*. Arlington, VA: ITAA, 1997.
19. Information Technology Association of America. *Help Wanted 1998: A Call for Collaborative Action for the New Millenium*. Arlington, VA: ITAA, 1998.
20. Norman Matloff. Now hiring! If you're young. *The New York Times*, January 26, 1998, page A19.
21. Norman Matloff. Debunking the myth of a desperate software labor shortage. Testimony to the U.S. House Judiciary Committee Subcommittee on Immigration. Presented April 21, 1998; updated February 21, 2000. <http://heather.cs.ucdavis.edu/itaa.real.html>.
22. Steve McConnell. Problem programmers. *IEEE Software*. March/April 1998. <http://www.construx.com/stevemcc/bp14.htm>
23. Steve McConnell. *After the Gold Rush: Creating a True Profession of Software Engineering*. Redmond, WA: Microsoft Press, 1999.
24. J. Paul Myers, Jr. and Henry M. Walker. The state of academic hiring in computer science: An interim review. *SIGCSE Bulletin*, December 1998.
25. The National Academies. Workforce needs in information technology: Project summary. Washington, DC: National Academy of Science, 1999. <http://www4.nationalacademies.org/cpsma/itwpublic2.nsf/>.
26. National Science Foundation and the Department of Education. *Science and Engineering Education for the 1980's and Beyond*. Washington, DC: U.S. Government Printing Office, October 1980.
27. Office of Technology Policy. *America's New Deficit: The Shortage of Information Technology Workers*. U.S. Department of Commerce, January 1998. <http://www.ta.doc.gov/reports/itsw/itsw.pdf>.

28. Office of Technology Policy. *The Digital Work Force: Building Infotech Skills at the Speed of Innovation*. U.S. Department of Commerce, June 1999. <http://www.ta.doc.gov/reports/itsw/Digital.pdf>.
29. Eric Roberts, John Lilly, and Bryan Rollins. Using undergraduates as teaching assistants in introductory programming courses: An update on the Stanford experience. *SIGCSE Bulletin*, March 1995.
30. Eric Roberts. Conserving the seed corn: Reflections on the academic hiring crisis. *SIGCSE Bulletin*, December 1999.
31. H. Sackman, W. J. Erikson, and E. E. Grant. Exploratory experimental studies comparing on-line and off-line programming performance. *Communications of the ACM*, January 1968.
32. United States General Accounting Office. Information technology: assessment of the Department of Commerce's report on workforce demand and supply. Report number GAO/HEHS-98-106. Washington, DC: General Accounting Office, March 1998.
33. Robin Wilson. Computer scientists flee academe for industry's greener pastures. *Chronicle of Higher Education*, September 24, 1999.

Association for Computing Machinery Education Board

Chair
Headquarters liaison
Vice-chair

Peter Denning
Fred Aronson
Richard LeBlanc

Robert Aiken
Lillian (Boots) Cassel
Gordon Davies
Marvin Israel
Eric Roberts

Standing committees

Accreditation
College Education
Pre-College Education
Two-Year College Curriculum
K–12 Initiative

John Impagliazzo
Russell Shackelford
Robert Cartwright
Karl Klee
Jenny House

Task forces

Computer Information Systems

John Gorgone/Gordon Davis

Representatives

Computing Sciences
Accreditation Board (CSAB)

Della Bonnette, Neal Coulter,
John Gorgone, Kenneth Martin

Institute for the Certification of
Computer Professionals (ICCP)

Joyce Currie Little,
Jan B. Wilson

<http://www.acm.org/education/>