The Y2K Problem:

Technological Risk and Professional Responsibility

Mark Manion
Drexel University
manionm@drexel.edu

William M. Evan
The Wharton School
University of Pennsylvania

The Y2K problem evokes two contrasting responses: technological optimism and technological pessimism. The optimist sees the Y2K problem as only a *potential* technological failure, mostly because s/he has boundless confidence that is can be solved by technology, through rigorous compliance procedures. The pessimist, on the other hand, argues that the problem is *actual*, not potential. Efforts to make computer networks Y2K compliant have already entailed massive costs to the U.S and the world. Some researchers even claim that the Y2K problem already constitutes the single most expensive technological failure known to mankind. Moreover, the Y2K problem might lead to the most expensive wave of litigation in human history.

This problem has prompted President Clinton to approve the *Year 2000 Information and Readiness Disclosure Act,* signed into effect October 19, 1997. In this Act, Congress made public the following facts:

(A) At least thousands but possibly millions of information technology computer systems, software programs, and semiconductors are not capable of recognizing certain dates in 1999 and after December 31, 1999, and will read dates in the year 2000 and thereafter as if those dates represent the year 1900 or thereafter or will fail to process those dates.

(B) The problem described in subparagraph (A) and resulting failures could incapacitate systems that are essential to the functioning of markets, commerce, consumer products, utilities, government, and safety and defense systems, in the United States and throughout the world.

(C) Reprogramming or replacing affected systems before the problem incapacitates essential systems is a matter of national and global interest.[1]

By removing the first two digits of the year, hundreds of thousands of computer programs that keep our economy stable are on the verge of a meltdown. This is truly ironic, because *without* computers and their associated communications systems, much of industry, commerce, transport and distribution, government, the military, health services, education and research, etc. would grind to a halt. Now, *because* of a computer malfunction, they may *all* grind to a halt. What will happen if we lose our electricity, telephones, access to banks and money, food distribution, water supply, automobile fuel for days, weeks, even months?

Given the widespread diffusion and complex interdependencies existing between companies and countries throughout the industrialized and developing worlds, the potential confusion resulting from this built-in technological failure could be truly global in nature. The OECD reports that there exist much dissimilarity between the various countries in terms of their Y2K compliance and readiness. Since the Y2K problem is a "systems" problem, viz., even if one country gets its house in order, countries that have not done so may still adversely affect it. In fact, severe economic consequences such as a major global recession are predicted, says Dr. Ed Yardeni, chief economist and managing director of Deutsche Bank Securities. Yardeni predicts, "It could be as bad or worse than the 1973-1974 global recession."[2] Yardeni is not alone in his prediction.[3]

Anticipation of the Problem

Our dependence upon computer and communication systems is growing at a rapid rate, but as society becomes more dependent on computers, we also become more vulnerable to computer malfunctions. Computer systems by their very nature are unreliable and unpredictable and society has yet to come to terms with the consequences. The Y2K problem, now looming menacingly, was, in fact, *anticipated, and hence completely avoidable*. This particular example of technological failure in not the result of so-called "unintended consequences" of technology[4]— this problem was foreseeable and fully anticipatible.

As early as 1984, an article appeared in *Computerworld* diagnosing the problem and calling for programmers and managers to take heed of the date conversion difficulties that would happen at the turn of the century.[5] Gillin, the editor of *Computerworld*, reported the findings of William Schoen, the programmer who first identified the year 2000 problem. He discovered the problem in 1983 while working at one of the Big Three automakers. As Schoen attests, data processing people had known about the risk as early as the 1970s, but, as Schoen puts it, "It's just that no one thought their codes would last that long."[6]

Schoen even designed a programming solution to the predicament, calling it the Charmar Correction, a cure for "the serious problem ignored by the entire data processing com-

munity." He then proceeded to create a consulting company, Charmar Enterprises, and went on a campaign to market his "correction" to the problem. However, he elicited only two sales for the Charmar Correction, and dissolved Charmar Enterprises in 1984. The sale price for the Charmar Correction was a mere $995.00. This is indeed ironic, given the millions, even tens of millions that many corporations and other organizations are now paying to "fix" theY2K problem. Schoen was not alone, however, in campaigning for attention to the date-field encoding problem. As early as 1960 Greg Hallmuch of the U.S. Bureau of Standards was raising the issue.[7] In 1967 Susan Jones, assistant director of the Department of Transportation was urging Congress to address the date conversion situation.[8]

In addition, many claims have come from the programming community that their urgings to top management of their corporations to address the potential problem were all ignored up until very recently. The question of responsibility arises: how could management have been so shortsighted in the first place? Even worse, one must be able to account for how, once they became aware of the problem, most major companies could respond so lethargically, as if in deep denial. In short, how could we have let it happen? The answer to this question is not as easy as it seems.

This essay constructs a complex, ramified argument through which these questions can be answered. To this end, this paper analyses 1) the hazards and costs of the Y2K problem, 2) the causes of the problem, and 3) the professionalization of computer programmers and software engineers.

The Hazards and Costs of Y2K.

There are at least three types of systems that are affected by the date conversion problem. They are: personal computers, mainframe computers and the software running on them, and embedded microprocessors. The scope of the problem is extensive, but the problems associated with embedded microchips is especially crucial, given their pervasive utilization in most, if not all, of our sociotechnical critical infrastructure and safety-critical systems. The potential hazards associated with Y2K non-compliance are serious and far-reaching.

Commercial risks and potential losses associated with mainframe dysfunction, as well as individual risks and losses associated with PCs pose serious harm to businesses and consumers. But the risks and dangers associated with embedded chips are especially critical, since they involve serious threats to our entire critical infrastructure, including all safety-critical, financial-critical, and health-critical systems. They are threats to our social and political structures.

The estimates for overall expenditures to correct the Y2K problem are increasing steadily. The current figures (October 1998) estimate US costs to be anywhere from $150 billion to 600 billion dollars.[9] Worldwide costs are estimated to total about $1 - 2 trillion.[10]

Litigation costs alone are estimated to be more than $1 trillion. Experts project that costs to most large corporations will average around $30-$50 million. With some 300 billion+ lines of code to inspect, and at a cost of $10, $15, sometimes $20 a line, once can see where the expense comes from. Some experts claim that the problem is going to set the IT industry back 30 years.[11] The statistics in Table 1 illustrate the general costs to fix the problem.

Table 1. General Costs to Fix The Y2K Problem[12]

Corporation	Y2K Budget	Lines of Code	People on Project
Atlantic Energy	$3.5M	25 M	7
Canadian Imperial Bank of Commerce	$150 M	75-100 M	250-300
C. R. Bard	$11 M	8 M	10
Merrill-Lynch	$200 M	170 M	300
Nabisco	$22 M	17 M	50-60
Union Pacific	$44 M	72 M	104

As one can see, the disruptive consequences of Y2K are enormous and hence the severity of the problem demands some sort of accountability for wrongdoing and responsibility for harm in such grievous circumstances. Congress, however, has recently short-circuited the need for accountability by succumbing to corporate pressures to enact legislation limiting liability for losses due to the Y2K problem.[13]

Causes of the Y2K Problem

The most common misconception about Y2K is that it is a single problem. Unfortunately, this perspective had created a commonly held belief that the "problem" is trivial, although widespread, and that a single solution is possible. In reality, the causes and the solutions are multifarious and complex. As one computer guru put it:

> It [Y2K]...was *perpetrated* by people who decided that what we did yesterday was good enough for today and did not look out for tomorrow and evaluate the inevitable consequence of cutting corners. It was *exacerbated* by people who scoffed at warnings and were in denial and irresponsible. It was turned into a *crisis* by people who left it to the last minute.[14]

This section discusses the complex set of reasons and causes at the root of the perpetuation, exacerbation and crisis construction of the Millennium bug. Although the causes are numerous, the interaction of three factors—technical, programming, and managerial—can be identified as a framework for analyzing the problem.

Technical factors

Lack of internationally accepted date standards. There exist no universal standard for date representation. Following the National Institute of Standards of Technology, the U.S. pro-

tocol is month-day-year, so January fourth, nineteen ninety-eight would be 01-04-98. Canadians and Britons reverse day and month so that same day it would be 04/01/98, or "April Fool's day. The Scandinavians use yet another system, putting the year first: 98.01.04. The International Organization for Standardization has as its standard: 1998:01:04. Notice that this standard includes the four-digit year. The lesson to learn is that standardization is crucial in computerization. The industry needs to universalize its standards of operation, as well as standardize and keep extensive records on date-field labeling, programming documentation, and record keeping.

High cost of computing. The second technical factor was simply due to the primitive state of computing technology in the beginning. In the days of Hollerith cards, computer space was at a premium and computer memory was very expensive. Given that programmers wanted to conserve computer memory and storage space, which at the time was extremely expensive, they ended up encoding calendar dates in a six-digit format mmddyy, rather than an eight-digit format. This equals a 40% saving for relatively no loss of information, and hence capital. This may account for the original decision for a six-digit date format.

Programming factors

Unexpected tenacity of original programs. Most programmers did not envision that the programs they wrote 30 years ago would still be running at the end of the 20th century. This permitted the development of psychological processes such as rationalization, dissociation, and other mechanisms of psychic numbing to avoid the cognitive dissonance between what they knew to be wrong, but their insistence on continuing their defective practice nevertheless.

Code Re-Use. A fourth complication is that virtually all new applications have algorithms from previous systems. The re-use of algorithms that have a hidden date-processing fault is one reason the Y2K problem is so extensive.[15] Incidentally, this fact is what likens Y2K to a virus: faulty algorithms get used and re-used, spreading their deadly payload to more and more systems.

Since successive applications are often built on earlier programs, or incorporate subroutines from other programs into their own structure, this means that successive applications are constructed on the basis of what could be faulty data.

Programs are still written using old algorithms. Even the best and most modern code in the world could be hamstrung by historical data that are faulty.[16] In fact, much software, which vendors and manufacturers knew were "infected" with the "millennium bug", have continued to be sold until very recently.[17]

Systems compatibility. Systems operating software, as well as customized programs, have been designed to be compatible with older versions. Out-of-date programs have supported leading edge replacement systems. Designing new systems to be compatible with older systems is generally a resourceful way to maximize efficiency. This feature may be consumer and producer friendly, but if this is done neglecting the values of quality, reliability, and science, the move to universal compatibility will inevitably lead to the design of faulty systems. Even when a generally good thing, systems compatibility allowed the "bug" to spread like a virus or bacteria. Moreover, the conversion problem was not addressed when designing systems for compatibility.

Managerial factors

Managerial accounting protocols. One major cause of the problem stems from the fact that accounting procedures have treated software expenditure to be an expense in the period incurred. This means that spending money on maintaining software has been treated like a telephone bill. It gets paid regularly, but at the end of the day it is perceived as not increasing a corporation's net worth. This means that capital expenditure for fixing the problem is seen as coming off the bottom line. It was difficult to convince a CEO or CFO that a $5 million corporate expense to solve the Y2K problem, in the early 1990s, was a "good thing to do." This accounts for the management inertia on this issue.[18] In other words, Y2K compliance has been a "tough sell."

How do you convince management to take on a multi-million-dollar project where the return investment is zero? It shows stubbornness, tenacity, and ignorance of other factors stemming from bureaucratic rationalism, efficiency, and capital accumulation.[19] These kinds of factors are the result of rigid organizational hierarchies – the typical obedience chain of positive and negative reinforcement. Left unchecked, this led to amoral functionalism and a sense of amoral rationality.

Decisional Inertia. Another cause of the problem is decisional inertia on the part of CEOs and CIOs. One cause of the indecisiveness is that many top managers have been deferring attention to the problem in the hope that a "silver bullet" may come along to solve the problem. But most experts acknowledge that a "silver bullet" is very unlikely to emerge. "There are hundreds of computer languages and a wide variety of systems" reports Kazim Isfahani, industry analyst at Giga Information Group, an IT consultancy group.[20] This is a perfect example of the false optimism of technology and progress. The belief that for every problem of technology, a solution is found with a technological "fix."

Another managerial cause of decisional inertia that led to failure to act in a timely manner is top management's general ignorance of management information systems. Corporate and governmental management certainly appreciated the benefits and results of computerization, but took little or no interest in understanding the complexity of information systems management. Hence one important lesson to learn from Y2K is that top management must understand infor-

mation technology. In order to limit the effect of such causes in the future, one must force CEOs and CIOs to be skilled and knowledgeable about technology that is the lifeblood of their corporation. Moreover, business schools must train their students how to manage organizations that depend on complex information technology systems.

As recently as a few years ago, programs were being written that did not take into account date changes and date fields in data processing. In fact, one survey reports that only 20 percent of America's biggest companies have devised a full-fledged strategy to deal with the problem.[21] Timely planning depends on whether managers were alert to the issue. Most were not because the information technology (IT) industry was either in denial, or negligent. Even when managers became aware of the problem, they also exhibited denial and neglect. A typical response from industry experts is, "I won't be in this position or this company in the year 2000. It's not *my* problem."[22] For example, as recently as December 4 1998, the *Wall Street Journal* quoted a corporate executive as stating, "This year 2000 stuff is *waayyyy* over done. It's complete, complete lunacy."[23]

Given the grave business, legal, and social risks and hazards caused by Y2K, and, given the elucidation of such a large set of causes as the seven identified and discussed above, it is not difficult to conclude that accountability for safe, reliable, and beneficial information technologies has been greatly eroded in the Y2K case. In the next section, we turn to a discussion of the role of professionalization in the control and management of potentially risky technologies such as computerization.

Professionalization of Software Engineering

In his 1914 book *Business: A Profession*, Brandeis identified two distinctive attributes of a profession: the mastery of a systematic body of theoretical and technical knowledge and the development and internalization of an ethic of service.[24] Although he did not persuade business to heed his advice about an ethic of service, his blandishments are as timely as ever with respect to the professionalization of software engineers and computer programmers. Well-established professions communicate their ethic of service through a code of ethics and a procedure for monitoring their members' compliance with the ethical principles enunciated in the code. Hence, one way to determine a profession's self-acknowledged ethic of service is to look at their code of ethics. When one looks at the code of ethics of computer programmers and software engineers, it would seem that many of the principles of the code are violated in the case of Y2K.

Take for example the code of ethics of the Association of Computing Machinery (ACM), one of the most established of computing professions. As a "General Moral Imperative," the code of ethics of the ACM implores its members to

> Contribute to society and human well being...minimize negative consequences of computing systems, including threats to health and safety.

Now, given the extent and extremity of the risks and causes of the Y2k problem discussed above, it is safe to assume that this code was violated.

Take also ACM General Moral Imperative 1.2, which states

> Avoid harm to others...the computing professional has the...responsibility to report any signs of systems dangers that might result in serious personal or social damage. If one's superiors do not act to curtail or mitigate such dangers, it may be necessary to "blow the whistle" to help correct the problem or reduce the risk."

Obviously, few, if any, computer professionals followed this code in the case of Y2K. Or, take ACM Specific Professional Responsibility 2.1:

> ...Strive to achieve the highest quality...in professional work...The computing professional must strive to achieve quality and to be cognizant of the serious negative consequences that may result from poor quality in a system."

And ACM Specific Professional Responsibility 2.5:

> ...Give comprehensive and thorough evaluations of computer systems and their risks...Computer professionals are in a position of special trust, and therefore have a special responsibility to provide objective, credible evaluations to employers, clients, users, and the public...any signs of danger from systems must be reported to those who have opportunity and/or responsibility to resolve them."[25]

Computer professionals are responsible for the effective development and operation of information systems. When major events are known to occur that pose significant risks or that will compromise the effective operation of these systems, such as the year 2000 date problem, computer technologists have a professional responsibility to alert management and take corrective action in a timely manner. As computer ethicist Helen Nissenbaum writes, "If any reasonable person fails to take precautions of which he is capable, and that any reasonable person with normal capacities would have taken in those circumstances, then he is not excused from blame merely because he did not *intend* the outcome."[26] Since computer professionals failed to take reasonable precautions to avoid the Y2K problem, they are collectively responsible for the predicament, even if they did not intend to cause such a problem. Nor could they have fulfilled their responsibilities simply by reporting the problem to top management without any further action. This belief is shared by members of the computing community itself and is expressed by Leon Kapplelman, who states that

> Potential fallout from Y2K problems will put our [the computing profession] credibility to the test of fire. Let's face it. Good intentions aside, computing professionals created the Y2K problem. Notwithstanding varying degrees of complicity by engineers, auditors, accountants, users, management, and others, the simple fact is that the code is broken and the code is the responsibility of the computing profession.[27]

Ed Yourdon, computer and Y2K consultant, makes explicit the connection between a profession's adherence to a code of ethics and its role in the mitigation of technical failures, specifically the Y2K problem, when he states that "if we computing professionals had insisted on following that code of ethics [i.e., the ACM code], we might have avoided the year 2000 problem altogether..."[28]

The professionalization of computer programmers and software engineers has been, in many ways a slow process. The first major step was a workshop on Software Engineering Ethics held at Carnegie Mellon University's Software Engineering Institute.[29] Many of the topics at the workshop were of a pedagogical nature – how best to institute ethical concerns into the computer science and software engineering curriculum. It was as early as 1975, however, that specialists were calling for the professionalization of computer programming.[30] Palmer's article stressed the benefits that could result from the licensing of computer programmers, both for working professionals and their clients, as well as the larger society that depends on computers and the software that runs them.

There are two opposing views as to whether computer programmers should be required to submit to a licensing procedure. One side states that the licensing of computer professionals is one way to achieve a heightened sense of accountability, responsibility, and knowledge in software development. The other side states that governmental regulation should not get involved because it will stifle the creativity and innovation that computer programmers are known for.[31]

Before computer programmers and software engineers can or cannot be held liable for their actions, however, they need to clearly establish themselves as a profession, and, as we have stated, the process has been slow. For instance, in the mid-70s the Special Interest Group on Software Engineering (SIGSOFT) was formed by the ACM, and the *ACM Software Engineering Notes* and *IEEE Transactions on Software Engineering* were first published. In 1993 the ACM and IEEE Computer Society established a Joint Steering Committee for the Establishment of Software Engineering as a Profession.[32]

Finally, in January of 1994, the IEEE Computer Society, in consultation with the ACM, drafted a Software Engineering Code of Ethics.[33] Again, as with the ACM Code, many of the principles of this proposed code were violated in the case of Y2K. For instance, Principle 1.08 states that the professional software engineer shall

> Ensure adequate documentation on any project on which they work, including a log of the problems discovered and solutions adopted.

If this would have been followed in the case of Y2K, the actual place of date-fields in complex programs, as well as code that is re-used, would have been identified and documented, thus alleviating the literally thousands of man-hours already spent on identifying, line-by-line, the two-digit date fields in order to make the program Y2K compliant.

Take for example principle 1.12 that states

> Whenever appropriate, delete outdated or flawed data.

If this had been heeded, programmers would have refused to use two digit date-fields long ago, and not continued to use them until very recently.

Another principle that was violated is 2.0 which states:

> Disclose to appropriate persons or authorities any actual or potential danger to the user, a third party, or the environment, they reasonably believe to be associated with the software or related documents for which they are responsible, or merely know about.

According to this principle, programmers have a responsibility to insist that top management take action in the event that software, which they write, poses a "potential danger" to the user or a third party. It is safe to say that Y2K poses more than potential danger to individuals, corporations, and society at large, given everything demonstrated above.

Principle 2.04 states that professional software engineers must

> Cooperate in efforts to address matters of grave public concern caused by software or related documents.

This principle obligates professionals to go beyond merely reporting on problems, but also to help "coordinate efforts" that cause "grave public concern." This, we can assume, would include the possibility of blowing the whistle on companies that refused to address the problem that Y2K has caused, even after the problem was brought to top management. In this sense, this principle is analogous to ACM's General Moral Imperative 1.2. This calls for professionals to recognize that their responsibilities go beyond the client, even self-interest, in the spirit of an ethic of public service.

To recognize the wider social responsibilities of the profession, principle 2.07 states

> [Do] not put self-interest, the interest of an employer, the interest of a client, or the interest of the user ahead of the public's interest.

Given that computers have a central and growing role in commerce, industry, government, medicine, education, social affairs, and private use, the need for computer professionals to heed their social responsibilities can be expected

by the public at large. Computer professionals *have* in fact realized the growing necessity of their important role in the society at large. In 1981, as part of the growing concern over the increasing use of computing technology in military applications, specifically, the perceived increased threat of nuclear war, a group of computer professionals organized what came to be known as the "Computer Professionals for Social Responsibility" (CPSR). Its concerns cover everything from military use of computing technology to issues of civil liberties in cyberspace. This model organization stands as a testament to the need for computer professionals to recognize the leadership role they can, and must take, if society is to be protected from the negative effects of computer technology.[34] In fact the Y2K problem has put into the question the professionalism of the computing community,[35] and it is incumbent on computer professionals to "save their sacred honor" by responding to Y2K in a quick and thorough fashion.

Conclusion

The need for a professional code of ethics arises mainly due to the unequal balance of power between two parties – the professional and the client. The professional has all of the expertise upon which the client is totally dependent. In the medical context, for example, the patient is vulnerable to the knowledge and directives of the doctor and must have utmost trust that the doctor will act exclusively in the best interest of the patient. Similar situations of vulnerability and trust exist between lawyers and their clients. Analogously, a vulnerable public can be harmed by technology created by engineers employed by corporations and governments that develop large-scale technological systems. One way to insure that doctors, lawyers, and engineers can be trusted to act in the interest of those they serve is to create in these professions a high level of commitment to an ethics of service. From the perspective of the dependent and vulnerable clients, it is essential that these professions scrupulously enforce their codes of ethics.

As society becomes more and more dependant on computing technology and information systems, it becomes more and more vulnerable to harm if these systems fail. Nothing illustrates this more than the Y2K problem. Hence, the conclusion of this essay is: the computing community should professionalize itself, by requiring their members to be licensed, and enforcing a code of ethics that mandates accountability on the part of their members.

Y2K has already turned out to be the single most expensive professional failure made in human history, and, no matter what else Y2K may bring, it constitutes a perfect case study in software engineering ethics, managerial ethics, and engineering ethics. ◆

Notes

[1] *Year 2000 Information and Readiness Disclosure Act.* Oct. 19, 1998, P.L. 105-271, 112 Stat. 2386.

[2] *Harvard Business Review*, 76 (4), July /Aug, 1998, p. 162

[3] Jimms, J (1998). Could Y2K cause a global recession? *Fortune* 138 (7) pp. 172-176.

[4] Tenner, E. (1996). *Why things bite back: technology and the revenge of unintended consequences.* New York: Vintage Books.

[5] Gillin, B. (1984). "The problem you may not know you have", Com*puterworld*, February 13.

[6] *Ibid*, p. 7.

[7] Munro, Neil (1998) "The big glitch," *National Journal* 30 (25) pp. 142-149.

[8] *Ibid.*

[9] Foremski, T. (1998, Wednesday, December 2). Millennium 'bomb' is already ticking. *Financial Times.* Information Technology, pp. 1.

[10] Gartner Group, 7 Oct. 1998. http://www.year200.com/costs.htm.

[11] Reinke, B. (1998, October). IT industry consultant. Quoted in Phillips, William. Here comes the millennium bug. *Popular Science*, 253 (4), p.92.

[12]. The statistics in this table were gathered from Computerworld 31 (51), December 22, 1997, pp. 2,5,6,8 and Computerworld 32 (25), June 22, 1998, pp. 7, 8, 10.

[13] Barr, Stephen (1999). "Deal reached on Bill to limit Y2K liability." *The Washington Post*, June 30, section A, p. A012.

[14] DeJager, Peter. (1998). "It's a people problem." Available at: http://www.year2000.com/archive/people.html

[15] Keough, J. (1997). Your safety net has big holes in it. *Solving the Year 2000 Problem.*Boston: AP Professional, Chapter 3.

[16] Fairweather, B. (1998). Not facing the future: computing chaos in the new century? http//www.ccsr.dmu.ac.uk/resources/professionalism/millennium/Y2Kprob.html

[17]. Miller, C.S. (1995). "Microsoft Wakes Up to the Problem.." *Infoworld*, 20 (15):1-3: Condon, Don. (1995) "Microsoft Tries to Rewrite Programming." *Infoword* 24 (6): 7-12.

[18] Garner, M. (1996) Why the year 2000 problem happened. http://www.is.ufl.edu/bawb080h.htm.

[19] Meall, L. (1995). The century's time bomb. *Accountancy*, 116 (128), 52-57.

[20] Foremski, T. (1998, Wednesday, December 2). Millennium 'bomb' is already ticking, *Financial Times*, pp.1; Cf., Newling, R. (1998, Wednesday, December 2). No magic bullet to save the laggards. *Financial Times*, pp. 8

[21] Peters, James. (1997) "If wishes were noises," *Computerworld* 31 (51), December 22, p. 2; Hicks, John (1997). "Many companies just starting to address the Y2K problem." *Byte* 21 (9), p. 24-28.; Feine, Jacob. (1997). "Slow responses to year 2000 problem." *IEEE Software* 14 (3), pp. 126-133.

[22] Furma, Jeff and Martola, Alberta (1994) "Year 2000 denial," *Computerworld* 28 (43), pp. 70-

[23] Binkley, Christina (1998). "Millennium bugged: the big Y2K problem is the silly questions," *Wall Street Journal*, December 4, p. 1.

[24] Brandeis, L. (1914) *Business: A Profession*, Boston: Small, Maynard, and Company.

[25] We are grateful to the paper by Cappell and Kappelman, Cappel, J. and Kappelman, L. (1998) "The Year 2000 Problem and Ethical Responsibility: A Call to Action, *Information Society* 14 (3), pp. 187-197 for drawing attention to the particular codes cited.

[26]. Nissenbaum, Helen, (1994) "Computing and Responsibility," *Communications of the ACM* 37 (1), pp. 77.

[27] Kappelman, Leon (1999) "Saving Our Sacred Honor," *Communications of the ACM* 42 (5), p. 23.

[28]. Yourdon, Ed. (1998) "The Moral Dimension of Y2K," *Computerworld* http://www.computerworld.com/home/print.nsf/all/98121482A6 (accessed 3/24/99).

[29] .Gotterbarn, Donald (1990) "A Workshop Report: Software Engineering Ethics," *Journal of Systems Software* 11 215-216.

[30]. Palmer, George (1975). "Programming, The Profession That Isn't," Datamation 21 (4), pp. 23-29.

[31]. Gotterbarn, Don and Webber, James (1994) "Can Computer Programmers Commit Malpractice?" *Computerworld*, 28 (35), pp. 37-41.

[32]. Bagert, D. (1999). "Taking the Lead in Licensing Software Engineers," *Communications of the ACM* 42 (4), pp. 27-29.

[33]. Gotterbarn, D., Miller, K. and Rogerson, S. (1997) "Software Engineering Code of Ethics," *Communications of the ACM* 40 (1), pp. 110-118.

[34]. To find out more information about CPSR, please visit their web-site at: http://www.cpsr.org

[35]. Pollack, Andrew (1999). "Year 2000 Problem Tests Professionalism of Programmers." The New York Times, May 3, section C, page 1.