Richard De George

COMPUTERS, ETHICS AND BUSINESS

Richard De George

The Year 2000 Problem (Y2K) has received a great deal of attention and commentary as the fateful date, January 1, 2000, approaches. Yet there has been no mention, to my knowledge, of any ethical considerations, despite all the reports, the worry about harm, the American Express Company's recall of its credit cards with "00" expiration dates because some outlets could not correctly process the date, and the law suits already being filed.[1] No one has made any moral charges or cast any moral blame, nor has anyone accepted any moral blame. Unfortunately the situation is not an anomalous one. For when it comes to computers and computer related activities, moral responsibility is in short supply.

The situation can be characterized as the Myth of Amoral Computer Programing and Information Processing. The Myth is expressed in language in which computers are the culprits, and of course, since computers are not moral beings, they can bear no moral responsibility. "It's the computer's fault" is an excuse often translated into "computer error," as if such explanations constitute legitimate excuses. The false hidden premise is that it is no one's fault if the error is the computer's. But computers do not make errors, despite the "error," "fatal error," and "illegal operation" messages with which computer users are familiar. The errors are the results of human mistakes, usually in either the writing of programs or the entering of data. Blaming the error on the computer is simply a way of refusing to accept responsibility for human mistakes or for human actions.

The Y2K problem is in some ways a uniquely computer problem. It has significant implications for business and for society as a whole, and is symptomatic of the extent to which the information society has integrated computers into everyday life and the extent to which we depend on computers. Most often everyday use takes place without the typical person realizing the extent of the dependence, the consequences of such dependence, and the degree to which human beings have abdicated responsibility for what they do or for what happens to them as a result of such abdication. The Y2K problems thus provides a microcosm of a variety of ethical issues both individual and collective or societal. I shall dissect and analyze a few of them.

The Y2K problem is the problem that all computer users--especially businesses and governments that use mainframe computers--face when the year 1999 ends and the year 2000 begins. The problem results from the fact that year dates have traditionally been entered into computers by only the last two digits, the initial "19" being assumed. The problem is that in the year 2000 entering the last two digits, or "00", will result in the computer assuming the first two digits are "19", causing the computer to read the date as 1900 instead of 2000, to report some default year (in early personal computers it is 1980), or to crash. Unless computers are fixed to handle the next century, assuming the first two digits of the year are "19" will yield untold difficulties as paychecks are not issued; people are declared no longer alive or non-existent because not yet born; insurance policies are not validated; bank interest is not paid; airline, hotel and other reservations are canceled; inventories are disrupted; security systems, elevators, and anything controlled by date-sensitive chips fail to operate; and records of all sorts are distorted in untold, imagined, and not yet imagined ways.

The scope of the Y2K Problem is enormous. The Gartner Group, an information technology consulting firm, estimates the cost of correcting the problem at from $300 to $600 billion dollars worldwide. It involves looking at 250 billion lines of computer code. The reason for the complexity is that dates in programs are placed into fields, and

that fields can be identified by arbitrary names and linked with other parts of a program or with other programs. The problem occurs in records, reports, program logic, data files, computer chips, and anywhere else a date appears for any reason or purpose. And date use is ubiquitous. There is no way of being sure of getting all the fields that contain dates without inspecting each line of code.

The way the problem arose is instructive. In the 1960s and 1970s all computers were mainframe computers. The computers were programmed in a programming language called COBOL (Common Business-Oriented Language). There were few rules about naming fields, and little in the way of formal or accepted programming procedures. Even more important, memory space was scarce and expensive. Hence any way of saving memory was a plus. Most commentators point to the shortage of memory as the reason programmers made their year fields for two characters rather than four characters. While this is true, if someone had foreseen the eventual cost of using two instead of four digits, programmers could have used four. The general convention in business, however, and not just in computers, was to use two digits for each of the day, month, and year. This saved room on forms, it saved typists' time in writing the date, and it was (and is)a generally accepted way of writing dates. Those who made and printed paper forms did not consider this a problem and reasonably assumed that people could properly interpret "00" as "2000." It is likely that because it was so general a practice, early programmers, who were interested in saving space, did not think of possible problems the convention might cause some thirty years later.

This is an explanation of why the convention developed and was followed. It says in effect that programmers were just like other people in using the convention and that they did not consider consequences thirty or more years away. Nor could they foresee the exponential growth of computer use. They in all likelihood did not imagine that the programs they wrote would be used and built on indefinitely and that they were writing in a sense for centuries.

There is no one we can point to who made the decision for a two place year field, no one we can identify who started the convention. It is difficult morally to fault any individual in those days for not seeing ahead. Yet we can legitimately raise the question: should computer programmers have seen ahead? If the answer is no, then are we faced with a situation in which technology simply develops with no one being responsible for being conscious or aware of its implications, with no one taking responsibility for it, and with no one being accountable for how it develops and for the harm that it does? If so, this is a greater problem than the Y2K Problem.

Somewhere along the line, as programmers built on previous programs and as they incorporated subroutines from other programs into their own, they must have realized that they were no longer sure of what a particular program contained or did not contain. It functioned as desired, but it was no longer the product of someone or some group that had mastery of the whole. Early programmers typically documented their programs. But documentation was often lost or ignored.

By the time some programmer or some manager discovered that they were no longer sure what their programs contained or how they were structured, the problem was that there might have been millions of lines of a program, and the cost of redoing it all from scratch would be enormous. At which point we begin to hold people responsible for what is in their programs and for what they sell or use is not entirely clear. But it is clear that now and for the next several years people are going to start holding others legally liable. Many predict that lawyers will have a bonanza in Y2K related law suits. What we will be faced with is the settlement of the issue in the courts, with perhaps some

legislation coming in if the situation, as it probably will, becomes too socially disruptive.

Thus the Y2K problem suggests four issues that deserve further consideration. The first is responsibility for programs. The second is the loss of control over the content of programs. The third is liability for harm done. The fourth is the interrelation of law, business, ethics, and computers.

1) Responsibility for programs.

Moral responsibility requires causal responsibility or connection with the events in question, knowledge of what one is doing, and consent to doing it. Moral responsibility can be mitigated or lessened if any of the three conditions are not satisfied. The conditions which mitigate responsibility are known as excusing conditions, and they may excuse one from responsibility to a greater or lesser extent.

Surely all programmers in the 1950s, 1960s, and 1970s knew that the year 2000 was coming and that assuming the first two digits of the date as "19" would be valid only up through the year 1999. Bob Bemer, who worked for IBM, foresaw the problem in the 1970s, and suggested that the year field be four digits rather than two.[2] Obviously he was ignored.

Early programmers cannot claim ignorance of the fact that the year 2000 was coming as an excusing condition. Nor does the fact that people customarily wrote dates using the last two digits of the year provide an excuse. Yet I have suggested that because of the expense of computer memory in those decades, and the fact that the early programmers could not foresee the development of cheap memory or the fact that their programs would be built upon instead of being replaced might provide some excuse and so some mitigation of moral blame.

If programmers in those decades could not foresee problems with the year 2000, programmers in the 1990s were certainly close enough to consider what would happen with the close of the old century. Clearly someone at some point not only recognized the problem but started to do something about it. Those closest to the problem had the responsibility to foresee difficulties and to report that something had to be done. Since the remedy for any firm that had been in business for more than a decade and used mainframe computers had the problem, all the companies should have been informed. It was then the responsibility of those with the authority to do something about the problem to take the appropriate action.

The likely scenario suggested by the Myth of Amoral Computer Programming and Information Processing is that the general managers, who were not computer programers and may have been barely computer literate, probably did not appreciate the enormity of the problem. If presented early by their Information Systems people with the very large projected cost of correcting it, the general managers perhaps understandably did not immediately authorize the expenditure of millions of dollars for what seemed at the time a far off problem. Most firms have more immediate problems with which to deal. Understanding this reaction, however, is not the same as exonerating from moral responsibility those who had it, or for their delaying fixing it sooner rather than later, and thus at lesser rather than greater expense to the firm.

It is generally accepted that those who produce harm are responsible for the harm they cause. Corporations that harm their customers are morally and usually legally responsible for making good on the harm caused. We can trace the causal link back, as lawyers are wont to do. In the case of the user of a product that contains a program that causes the product not to operate as normally expected, the customer has recourse to the supplier of the product. If the product contained a program that is defective in some way, the producer may be the developer of the product or may simply have purchased

or licensed the product or had it developed by a subcontractor. Responsibility for the program devolves then on the producer of the program. Programmers who work for an employer are responsible to the employer, but the employer owns their products as "work for hire" and so is responsible for the use to which it is put.

Although the courts may have the task or sorting this out legally, ethically each company bears responsibility for its products and for the harm it does by failures due to its products. We can imagine scenarios with respect to the Y2K problem in which some people are not paid on time because of computer glitches. Simply pointing to computer glitches is no excuse for the harm done one's employees. The company ethically bears the burden of making good the harm by either instituting loans or paying interest on the late checks or making sure that employees' mortgages are not foreclosed or utilities disconnected because of its doings. Whether this is legally enforceable or not, and whether individual workers can afford the time and expense of suing is doubtful. Nonetheless the moral responsibility remains.

The government bears the same responsibility in this regard for the harm it does as do corporations and businesses. The estimated cost for the US Government agencies to fix their Y2K problem is $20 to $30 billion according to the General Accounting Office.[3] Whether the Government accepts this responsibility if welfare checks are delayed, payrolls fouled up, or records unavailable remains to be seen. Nonetheless, the Government clearly has the moral responsibility for what it does, just as do other organizations and corporations.

We can generalize beyond the Y2K problem. Those who produce or incorporate programs into products are responsible for those products and programs, just as they are responsible for other products or goods they sell. Yet there is a tendency which we have noted in the Myth of Amoral Computer Programming and Information Processing for companies to disown responsibility for computer malfunctions or breakdowns, and for commercial software producers to issue disclaimers with their products claiming that by opening the product the user relieves them of all responsibility. That this has been accepted without much complaint by the general public is at best puzzling. One result has been for software producers to release their products before they are ready. Savvy software users know better than to purchase the first version of any new software product. Users have learned that instead of the extensive testing that should be done before a product is released, producers release a product which they know still has defects, and which they correct as the defects are reported to them. The general public thus provides some of the testing the producer should have done. Yet the buyer is not informed of this service to the producer, or paid for it if he or she reports a difficulty; nor does the product cost less because it has not been completely debugged when marketed. All of this is contrary to the general policy with respect to other products.

What has happened in these cases as with most other ethical issues related to computers is that the ethical dimension has been preempted by the legal dimension, and the laws have tended to reflect the business interests of the providers of computer programs and services.

Even in the 1990s, instead of changing all the old two digits fields to four digits, many companies and programmers have decided to stick with a two digit field and rely on some fix such as to treat "00" as greater than "99" in fields dealing with years, or treating all dates lower than some number, e.g., "20" as being in the 21st century instead of the 20th, a solution that will be good only until the year 2020 approaches. For some companies this is ethically responsible. For others it is not, and those responsible are simply shifting the problem forward, when it will be harder to fix.

There can be no excuse for programmers to use two digits rather than four for years in new programs; yet many still do, using some algorithm or other to keep the two centuries straight and assuming that there will never be need for more than two centuries, and that their programs will not be in use by the time the algorithm no longer works because of the next century. A lesson to be learned from the Y2K Problem is that no one presently knows how long programs that are being written today will be imbedded in programs used many years into the future, and that programmers have the moral responsibility to avoid problems that can be avoided, even if the problems are foreseeable only for the distant future.

2) Loss of control over the content of programs.

The Y2K problem points to a larger and potentially more significant problem. With the thirty to forty year experience we have with computers and computer programs thus far, the Y2K problem demonstrates the extent to which we as a society, government, and businesses, as well as individual users, are losing control over the programs that we use and have come to rely upon.

The Y2K problem arose in many cases because of early programming, which was often idiosyncratic in labeling and documentation. There were few imposed and widely recognized standards, since the standards had yet to be developed. Most of the programming was done in COBOL, which was widely taught in colleges, but which has long since been replaced by more advanced programming languages. Hence to correct the Y2K problem step one was to find people who knew COBOL and could go back and read the old lines of instruction. The number of people proficient in COBOL was and is comparatively small, and a large number of those working on the problems are people who had retired and have been lured out of retirement by the high pay people with such knowledge are commanding. Firms often find that the people they hire are soon discovered by other firms needing people skilled in COBOL, and they lose their employees to competitors who pay more.

It does not take much imagination to see what would happen if a similar problem arose twenty years from now. The number of people skilled in COBOL would by then have shrunk to a very small number. Eventually the language will be unknown by any but perhaps historians of computer languages. By continuing to rely on old programs instead of rewriting them, society as a whole runs the risk of eventually using and relying on programs that no one can fix, and that no one can even examine knowledgeably. Computers will be black boxes with output that one takes on faith without any experts to guarantee that what goes on within them is reliable. Nor is the problem only with COBOL. The life-span of computer languages is already incredibly brief. As programmers continuously incorporate older programming code into new programs or as they build on existing programs, it is not hard to foresee that society in general as well as governments, individual firms, and organizations will be relying on code that no one can any longer read.

Because programs now often involve millions of line of code, it is not possible for any single individual to write or rewrite it all. Nor would that be of particular use, since that person would then be the only one with command of the whole.

With loss of control there is a tendency to disclaim responsibility. If unforeseen and untoward events occur, they are blamed on the computer, which is to say that no blame is assigned or assumed. Unforeseen computer events become unforeseeable computer events, which take on the status of acts of God. Only in this case God is the computer. Acts of God are events that typically are excluded from insurance policies, although one can insure against certain specific damages, such as that caused by flood or earthquake.

Insurance companies might similarly start issuing computer damage insurance, or alternatively they might start excluding such harm from their umbrella or specific policies. This scenario accepts the current trend towards lack of control and lack of responsibility and accountability as inevitable. Such an attitude reinforces the cause and provides no incentive to find a way to reverse or push back or stop the loss of control. If no one is responsible for doing anything along these lines, no one will do anything to change present procedures or attitudes.

The Myth of Amoral Programming and Information Processing is not a solution but the heart of the problem.

Critics have for a long time warned about technology having an impetus of its own that tends to develop and go its own way without human guidance. What can technically be done is sooner or later done, be it developing weapons capable of wiping out humanity or biological advances in reproduction, from cloning to cross fertilization across species, including the human species. Yet in some areas people and societies have set limits and at least tried to apply moral as well as legal sanctions. In the area of computers this has not yet happened, and the trend seems to be in the other direction.

A start in the right direction would be to hold companies, and for companies to hold providers and consultants, responsible for their programs and the untoward results they produce. Exactly how programmers would control for their use of old code and how far they would be expected to foresee uses to which their programs would be put are questions that should be discussed. The present problem is that they are not presently raised, much less discussed. What documentation would make a programmer's code accessible twenty or fifty years from now? As new languages are developed, should translation programs be produced that translate all the old code into the new language and search for possible glitches at that time? A translation code for COBOL already exists. If translation programs are required before a new programming language is adopted, and if the translation of old code into the new language is required, such procedures would slow up the transition from one language to another. But they would ensure that programmers could read all the code for which they are responsible, and could write programs to check what needs checking, since manual line by line checking, such as the Y2K problem presently demands, may be impossible as the string of code grows from the millions to the hundred of millions or even billions.

Accountability for what one does is the only way to provide a break on computers outpacing human beings and on humans blindly relying on computers to do what humans want. Without such accountability the result will be society's relying on computers to do what computers can do. Unknown presuppositions built into programs can emerge in perhaps strange ways, precluding lines of development or of imagination or of thought. Progress in knowledge often comes from challenging presuppositions. But one cannot challenge unknown presuppositions buried in programs. Some mathematicians worry about the reliability of mathematical proofs that can only be carried out by computers and cannot be checked independently of computers. How can we be sure, they argue, that there is not an error somewhere, and that the proof is indeed reliable? The situation with respect to such mathematical proofs is multiplied as we find the situation duplicated in field after field and area after area in which there is reliance on computer programs over which no one has control, which no one completely understands, for which no one can completely vouch, and for which no one is responsible.

If the argument is persuasive, then it is not too late to start enforcing both ethical responsibility and legal liability. A start would be to hold commercial programmers

accountable for their programs, such that they work as advertised and that they do not crash or make it necessary to reinstall Windows or whatever operating system one is using. Such products waste the users' time and are enormously frustrating. Yet there is no penalty for such action. If the doctrine of truth in advertising and a concept comparable to punitive damages that could be invoked without an expensive lawsuit were instituted, it would be a signal that software vendors are responsible for their products.

The problem of accountability for large programs that are individually programmed for a specific need is more complicated. Nonetheless the principle of accountability is the same. In areas of great potential danger--such as national defense or nuclear reactor stations--redundancy should be built in and there should be moral pressure and legal sanctions to provide the needed incentive not to rely on computer black boxes and to maintain continuing control even as computer programs grow larger and larger. To give up at any stage is to condemn the further development to possible computer anarchy.

3) Liability for harm done.

The U.S. Federal Government has developed "Recommended Year 2000 Contract Language" for commercial supply products to ensure they are year 2000 compliant. The language covers hardware, software, and firmware. The recommended clause simply states that "the remedies available shall include repair or replacement of any listed product."[4] That warranty seems obvious enough, yet is explicitly stated to offset any claims made by software or hardware vendors that disclaim responsibility for any defects or harm, including loss of profits (for businesses) or other consequential damage. Although the clause provides an incentive for providers to make sure that their products are year 2000 compliant, the penalty is far from severe. There is nothing that stops a provider from releasing a program or product which it is not certain is year 2000 compliant, with the assurance that in the meantime it can continue to work out possible bugs. The greatest penalty it suffers is replacing the item at a later date, if need be.

This is typical of warranties on software, which usually state they are at most limited to replacement or refund under certain conditions. Nor do many even claim that they are liable for such replacement even if the product does not perform for some specific purpose. Responsibility for fixing its programs to be year 2000 compliant is clearly the responsibility of each company that uses computers and of each provider of programs or date sensitive chips. Correcting the problem, moreover, is a multi phase process. The first phase, which I've mentioned, is locating all the places in which a date field is present. The second phase is changing the date field in such a way that the change is compatible with the rest of the program or system with which that piece of the program interacts. The third phase is testing the whole system to make sure it works before the crucial dates arrive. The fourth phase is implementing the corrected, tested program.

For reservation services that take reservations a year in advance, their systems must be operative by January 1, 1999. For companies whose fiscal years begin before the calendar year, it will be whenever fiscal year 2000 begins. For most other systems it will be January 1, 2000. No matter when the crucial date arrives, the moral responsibility of all companies is to have done the necessary testing in advance and to ensure that they cause their customers no problems. Ignorance is not an excusing condition. Moreover, not only must information systems departments have to clean up code for a company's mainframes, but individual managers must also be sure that they and their subordinates who generate data bases, documents, and programs correct their own work and make sure that what they generate is year 2000 compliant. There has been little moral pressure on companies to force them to act, nor has there been any legislation clarifying

corporate and governmental responsibility or providing incentives for companies to guarantee they will be year 2000 compliant. The unfortunate yet foreseeable result will be massive litigation, with judges determining liability.

What liability is appropriate for software? From an ethical point of view it would appear to be liability for harm caused by the product. This is offset by the conditions that one accepts when purchasing the product. If one agrees to buy a product "as is," then one accepts it as is. An implied contract for most products is that they will perform as advertised and that they will come up to the state of the art for that product. Cars should not burst into flame when hit from the rear at ten miles per hour. Lamps should not give one a shock when plugged in. Programs should function as expected.

When it comes to software, software manufacturers have attempted to exonerate themselves of responsibility and liability by including agreements with their products that state that if one breaks the shrink wrapped seal, one accepts the terms, conditions, and limitations set by the manufacturer. Those are the only terms under which the product is sold. They are thus not responsible if the product causes the computer to crash frequently, or if the program malfunctions periodically, or if it causes errors. Society and the general public have quietly accepted these conditions without question or protest. Yet they were not negotiated between buyer and seller, and they have not been given any ethical justification or rationale. The Myth of Amoral Computing and Information Processing simply covers over any ethical questions or issues, and illegitimately makes them illegitimate.

Strict liability for products in law is justified on three grounds. First, since harm is done to the consumer, the entity in the best position to make good on it is the producer. The producer can add a small charge to each of its products to cover its potential liability, thus insuring itself against the loss and spreading the cost among all its users. This is preferable to any small number of users suffering serious loss.

Second the knowledge that it will be held strictly liable gives the producer a strong incentive to make its product as safe and reliable as it can.

Third, the producer knows its product best and so is in the best position to foresee and correct as many malfunctions and misuses of the product as possible.

Comparable arguments apply to software products. Since we have already argued that the software producers should be held morally responsible for their products, holding them legally liable is appropriate and provides the incentive the producers now do not have to live up to their responsibilities. What the suppliers are morally and should be legally liable for depends in part on the product. A program that is embedded in a chip in a product and is not the product itself can result in whatever harm the product can result in. Just as the producer of the product can be held liable for the harm done by the product, it (or the subcontractor who produces the program) can be held liable for the harm done, unless the producer of the total product is willing to assume that liability. If the device is a health sensitive one, the harm may be considerable. If it is in a machine, the danger varies according to the machine. If it is a software program that runs credit card machines the damage is unlikely to be physical and will be the costs involved with a computer failure or malfunction. If it is in a program, such as a word processing or spreadsheet program, that crashes, for instance, because of the Y2K problem, at least a free patch to fix the problem is appropriate. But manufacturers of such products should not wait for the crashes and should make them available well before the event, preventing foreseeable harm. Crashes that result from failure to debug programs are often more annoying than dangerous for the average individual user. Once again, software manufacturers have disclaimed responsibility and liability for

these products. Some provide on-line help with difficulties, some do not. But the numbers one calls are rarely toll free and the cost of reporting a problem and receiving a remedy or fix is borne in large part by the user. This is not acceptance of responsibility. It is another aspect of the Myth of Amoral Computer Programming and Information Processing.

4) The interrelation of law, business, ethics and computers.

The interrelation of law, business, ethics and computers is exemplified by the Y2K problem. Ethics has taken a back seat to law as the method of assigning responsibility and liability, as well as in the area of ownership and even privacy. Law, in turn, has been driven primarily by the vested interests of business, rather than by consideration of or discussion of the common good or of computer users.

Originally software was primarily shared freely among the rather small number of people capable of and interested in developing software applications. Programs were made available and programmers built on each other's work freely and imaginatively. The general rule was that software was freely available unless otherwise noted. The presumption only later became that one should presume software is proprietary, unless it is explicitly declared as free or as shareware. The latter refers to programs that are copyrighted, but made available without cost on a trial basis. If one likes the program and decides to use it, then one is expected to send the originator a fee, usually nominal. Programmers preferred the presumption in favor of freedom, rather than in favor of ownership. Businesses preferred the latter, and business carried the day with legislators.

The method of protection that software secured was copyright, even though the fit was far from comfortable or appropriate. Originally, only the source code could be copyrighted. Even though the source code was translated via a compiler into the object code of zeros and ones that the machine could read, the latter was not protectable by copyright. Eventually that was changed. Copyright, established originally to protect the written word, and which had been extended to cover pictures, records, motion pictures and other works, was extended to software.

Some protection was appropriate, since those who had put their time and effort into writing programs should be able to protect their investments and they should be able to reap financial rewards. Without some protection, others might simply copy the program and sell it, reaping the rewards without the effort and depriving the original author of those rewards. The protection against pirating should be available in some form, and is ethically appropriate. The argument that all software should by right be free is defective in this regard.

Nonetheless, copyright was interpreted with respect to software in ways that are counter intuitive. When individuals buy software, such as a word processing program, they are not buying an object, such as a book. They are buying a license to use the program. The program enables them to do something, namely, write whatever text they wish. Yet the buyer, by analogy with other things they buy, believe they buy the right to use the program, destroy their copy, lend it, use it on more than one computer if they have more, and so on. They understand that they may not copy and sell it, anymore than they may copy and sell copies of books they buy. They understood they have no right to distribute the program freely to all, e.g., by making it available on the Internet. Yet, there is a large discrepancy between what commercial software sellers have had written into law and what a great many individuals do and believe with some justification is ethically permissible.

Lending programs to a friend is not in itself unethical, nor is it unethical in itself to use one's program on more than one computer if one has more than one. It is unethical

only because it is illegal and there is a general moral injunction to obey laws that are not in themselves immoral. Yet in this regard software providers have encouraged unreasonable protection. Software programs are licensed to corporations, which pay a basic fee and then a small fee for each computer on which the program is used. But such licenses are not available for individual users, who are expected to buy two programs at full consumer prices if they expect that the program will be used on their two computers at the same time. Some licenses allow placing the program on two of one's own computers if they will not be used at the same time--such as on a desktop and a portable, used by one person. One's family members may not use the desktop's program if the purchaser of the program is using it on the portable. The rules are not derived from common sense or from ethical considerations, but from a Procrustean copyright system that does not fit the medium it is protecting.

Copyright lasts for the life of the author plus fifty years, or seventy-five years for a corporate author. Most programs of any popularity are obsolete within three to five years, and are replaced by later versions. Yet one cannot legally give even one's obsolete versions of a program to a friend to use. It is not surprising that the rules are often breeched, especially since they cannot be policed.

Interestingly enough, in the 1980's, as personal computers started becoming popular, some software developers installed keys in the programs that prevented them from being copied more than twice--once as a backup and once on the computer. Those programs that were written in this way quickly fell into disfavor, while good programs which were shared became more and more popular. They sold well because they were good, and people often found out they were good by trying them first. This is the principle on which shareware is based. Those who paid for their copy of the software received documentation (usually a book, sometimes, a rather large one), as well as the right to call the company for technical support in case they ran into difficulties. Those who did not buy the programs, of course, received no support or documentation. Hence there was an incentive to buy the product, and to buy the newer versions as they became available. The success of companies such as Microsoft and the developers of programs such as Lotus and WordPerfect show that the practice by individuals that are technically illegal have not done great damage to the developers of good commercially marketed programs. If we apply the earlier argument about responsibility here, software companies would be held responsible for damage done to the purchaser of the program and legal users thereof, but not to secondary users.

The Y2K Problem also suggests the question of the length of time protection should be available for programs. Under copyright, the time for a corporate product is seventy-five years. Under a patent, the time is seventeen years. Commercial programs become more or less outmoded in three to five years. Yet the Y2K Problem teaches us that old code is often the basis for new code. Program developers may desire the protection of old code for the reason that, although old, it is embedded in the current program. Yet this argument seems poorly founded because it is highly unlikely any developer of a new program would go back and copy the old code in the old language rather than develop new code. Seventy-five years of copyright protection with respect to mainframe programs similarly seems misplaced. First of all such programs are not generally available. Second it is dubious that old code in such programs would be attractive to developers of new programs, especially since it is so difficult to guarantee compatibility in large programs that mix different languages and that come from different times in which different assumptions may have been operative and not documented.

To return to the issue of responsibility, the guiding principle should be that those

who have control and ownership have correlative responsibility. If a company claims complete ownership rights, it should carry complete responsibility for harm done by its product.

The development of computers and of computer programs has been rapid. It has been too rapid for the general public to completely grasp what is at stake and for society as a whole to develop its moral intuitions about many aspects of the development. Corporations that develop the products have been in the forefront of seeking and getting protection, and the protection has come about by using the instruments of copyright and patent which are poorly suited to handle the new kinds and forms of property. In these developments ethics has rarely played a role or been considered. It is time for the ethical considerations to emerge and for the Myth of Amoral Computer Programming and Information Processing to be unmasked the way the Myth of Amoral Business[5] has been unmasked. Corporations are now willing to discuss and accept moral responsibility for their business actions in a way that was exceedingly rare in the 1960s. A similar attitudinal change is a pressing need as computer technology and information systems impact more and more directly and centrally on the lives of all in contemporary society.

University of Kansas

NOTES

[1] The first suit filed was by Produce Palace International, Inc., which claimed that its cash registers, which it purchased in 1995, crashed whenever credit cards with expiration dates beyond 1999 were entered into the register's card reader. If the claim, which is disputed by the manufacturer, is sustained, Produce Palace is certainly correct in expecting that $150,000 cash registers purchased in 1995 should be able to handle transactions involving credit cards with expiration dates in the year 2000. (Christopher Simon, "Lawyers See Dollars in Computer 2000 Ills," *The Wall Street Journal*, November 6, 1997, p. B21.)

[2] Lynda Radosevich, "Millennium bug already taking its toll," *Info World*, Jan 12, 1998, vol. 20, 2, p. 19.

[3] *USA Today*, Del James, "Year 2000 problem makes experience pay off," March 21, 1997, B1.

[4] "Year 200 Contract Protection and Warranty Language" on http://www.itpolicy.gsa.gov/mks/yr2000/contlang.htm.

[5] For a discussion of the Myth of Amoral Business, see Richard T. De George, *Business Ethics*, 4th ed., Englewood Cliffs: Prentice Hall, 1995, pp 5-7.