

Who Owns Information?

FROM PRIVACY TO
PUBLIC ACCESS

Anne Wells Branscomb



BasicBooks

A Division of HarperCollinsPublishers

FOI
ACI

Copyright © 1994 by BasicBooks,
A Division of HarperCollins Publishers, Inc.

All rights reserved. Printed in the United States of America. No part of this book may be reproduced in any manner whatsoever without written permission except in the case of brief quotations embodied in critical articles and reviews. For information, address BasicBooks, 10 East 53rd Street, New York, NY 10022-5299.

Designed by Ellen Levine

Library of Congress Cataloging-in-Publication Data
Branscomb, Anne W.
Who owns information? : from privacy to public access / Anne Wells Branscomb.

p. cm.
Includes bibliographical references and index.
ISBN 0-465-09175-X

1. Intellectual property—United States. 2. Privacy, Right of—United States. 3. Freedom of information—United States. 4. Data protection—Law and legislation—United States. I. Title.

KF2979.B67 1994
342.73'0853—dc20 93-44348
[347.302853] CIP

94 95 96 97 ♦/HC 98 7 6 5 4 3 2

1
2
3
4
5
6
7
8
9
10
NO
INI

8

Who Owns Computer Software?

In 1987 an irate reader wrote a letter to the editor of *Computerworld* expressing his frustration over litigation initiated by the Lotus Development Corporation. Lotus had gone into court claiming copyright infringement of its highly successful spreadsheet program Lotus 1-2-3. The defendants were several software companies offering "clones," look-alike programs that purported to be able to do precisely what Lotus 1-2-3 did, but at a lower initial cost. The reader wrote: "Lotus Development Corp.'s suits against 1-2-3 look-alikes may make good business sense but are little more than attempts to pull up the gangplank now that they are on board the ship. The problem is that they are trying to make it appear that their products are islands in a sea of unrelated personal computer products."¹

The writer was by no means alone. Computer scientists at the prestigious Massachusetts Institute of Technology thought the issue important enough to picket the Lotus offices in Cambridge, Massachusetts. The picketing was led by two giants in the field: Marvin Minsky, founder of MIT's Artificial Intelligence Laboratories and a professor of electrical engineering, and Richard Stallman, developer of EMACS, a popular programming editor. In a style harking back to 1960s protests, the group chanted: "Put your lawyers in their place; no one owns the interface." A spokesperson for Lotus called the protest "silly," and said it represented an "ignorance of the realities of the marketplace."²

At one level, the suit was nothing more than one more step in the legal process refining exactly what aspects of software programs are entitled to copyright protection. In this case, Lotus claimed that three components of its program were so entitled: its interface, command structures, and source codes. But at a second level, the strong public response to Lotus's legal action suggests that thirty or so years into the information revolution, the computer community was not yet near a

consensus on the protection to which software developers were entitled.

As this chapter will suggest, the tension in such cases is not solely between the interests of the two litigants but also between rights of the party seeking copyright protection and the needs of the society at large. This tension is expressed in the following question: Are certain elements of information technologies so critical to the continued evolution of the information revolution that our courts must think very carefully before agreeing to privatize and protect them, even where a clear case can be made that such protection had been extended in the past to noninformation products?

The story of three electronic spreadsheets—VisiCalc, Lotus 1-2-3, and Quattro Pro—and the efforts of their owners to protect them are instructive in this regard.

The world first heard about electronic spreadsheets in September 1979. An innocuous ad on page 51 of *Byte* magazine announced a new software product called VisiCalc and suggested that it could dramatically compress the time required for managing data. All the user had to do was enter numerical information in appropriate boxes and the program would automatically update all important calculations affected by these figures. VisiCalc was devised by two MIT graduates, Dan Bricklin and Bob Frankston. Bricklin got the idea from his calculator at the Harvard Business School: "I had my calculator, ran my numbers at home at night, and when I made errors, daydreamed about 'word processing' numbers so that I could recalculate them with a new assumption—say, 12 percent instead of 10 percent. My image was based on a calculator with a mouse and a heads-up display like a fighter plane so you could see the numbers."³

VisiCalc did not impress its early reviewers. One of Bricklin's Harvard Business School professors scoffed at it. Still, Frankston and Bricklin pushed on. They took their program to a fellow student, Dan Fylstra, who was selling game cassettes. Fylstra agreed to distribute the product through VisiCorp, a firm he set up on the West Coast.⁴ He then took a prototype to Apple chairman Mike Markkula, who was less than enthusiastic. He was more interested in demonstrating his own checkbook-balancing program to Fylstra than in hearing Fylstra out about the benefits of VisiCalc.⁵

He was also less than prescient about the future of the electronic spreadsheet. Because of VisiCalc, one of his Burlington, Massachusetts, dealers was to become the most successful Apple dealer in the country at that time. The first copies of the program were shipped in October 1979. The developers of this new phenomenon were apprehensive about

calling the new product a *spreadsheet*, the popular name by which such programs have come to be known.⁶ The name VisiCalc was coined from its concept as a visible calculator.⁷

VisiCalc hit the market in a big way. In the annals of computer history, it will be remembered for its impact not only on the nascent software industry but on the nascent personal computer industry as well. As a software breakthrough, VisiCalc gave birth to the "what-if" kind of computer considerations,⁸ allowing those responsible for business planning to compare a series of contingencies with an ease and facility never before possible. VisiCalc made important contributions to the personal computer industry in two major ways. First, by empowering the personal computer with forecasting capabilities, it pulled back planning functions from the people working on mainframes. Second, by giving a large new group of potential computer purchasers an easily understood reason to take the plunge, it dramatically expanded the overall market for personal computers. According to Frank Rose, a software analyst: "Before VisiCalc, personal computers were toys—fun for hackers who liked to diddle with electronics but not suited to serious business. VisiCalc . . . was the product that ushered the personal computer into the American office. In the process, it wrested power away from the data-processing managers who controlled corporate mainframes."⁹

VisiCalc also helped jump-start the IBM-PC.¹⁰ Given the fact that many people bought their first computer solely to access this new program, sales of all personal computers—Apple, its competitors, the IBM-PC, as well as its many clones—skyrocketed.

Million-dollar fortunes were made—and then lost—as the product soared in popularity and then sank into obscurity. For some unfathomable reason, possibly the euphoria of surfing on their wave of success, the developers of VisiCalc let an employee, a young Beta test manager named Mitchell Kapor, depart the company with a clearly established legal right to produce his own software that VisiCorp had declined to develop and market. Thus they contributed to their own demise. By 1984, one year after Lotus 1-2-3 was introduced, the price of VisiCorp stock plummeted from its high of eleven dollars a share to twenty cents a share. Over time Lotus 1-2-3 made millions for Kapor and launched the start-up of one of the industry's most productive and successful software companies, Lotus Development Corporation. Eventually Lotus acquired what remained of its predecessor, laying to rest the question of whether any ill-acquired proprietary information might have been transported away from the company.¹¹

The fall of VisiCalc—intertwined as it is with the rise of the program that brought it down, Lotus 1-2-3—offers an interesting case study of

the proprietary issue in computer software, an issue whose definition would lead to a complex series of court cases. Fortunately, or unfortunately, depending upon your perspective, VisiCalc was developed during the period in which the patent office did not favor patents for software, and it was not unusual for lawyers to steer their clients away from this alternative form of protection for spreadsheet inventions.

Nor was it clear whether copyright law applied to computer software, although the copyright office did accept software for registration with the understanding that there was some question about the legality of providing software copyright protection. Bricklin admitted that he and Frankston would have sought a patent had they thought it possible.¹² If they had, the owners of VisiCalc would have acquired a seventeen-year monopoly (others would have called it a stranglehold) on the development of such spreadsheets; competitors wanting to enter the field would have had to seek a license from them. Later events suggest that this would have severely retarded developments in the spreadsheet field.

Protection of Software Development

Part of the reason there is no certainty concerning legal protection of computer software is historical.¹³ When computers first appeared, in the 1950s, the software (the instructions that tell the computer what to do) was sold as a part of the total package of equipment, including installation and maintenance of an operating system. Little attention was paid to the separate value of the software at this early state of development. Many thought that the big powerful machines were so expensive and so specialized in their largely computational function that the industry would consist of maybe a dozen manufacturers at most. As long as you received software when you purchased the hardware, few envisioned a day when software sales would develop into a stand-alone business. Indeed, in those days the operating software was included as a part of a computer's patent.

In the late 1960s, when computer manufacturers were required by the antitrust laws to "unbundle" their software and offer it separately from the hardware, lawyers were not at all confident of how legal rights of ownership and control should be asserted. At first, many lawyers expected the patent law to be an appropriate haven for inventors of computer programs to obtain a monopoly over their innovative work (truly new, original, and nonobvious) for enough time to allow recovery of start-up costs. The patent office shied away from issuing patents, however, in part because of the administrative burden. The copyright office

(where only a modicum of originality is needed for protection) was more hospitable, accepting registrations subject to verification by the courts or by Congress that they were within their mandated authority in doing so. At the beginning, therefore, it was not at all certain that software would find protection in patents or in copyrights. So lawyers turned increasingly to the trade secrets law—a body of law designed to permit businesses to protect information that gives them a competitive advantage—but it is an alternative that requires the cooperation of the recipient in keeping the secrecy.

In the early 1980s three things happened that changed the picture drastically. In 1980 Congress enacted an amendment to the copyright law legitimating software copyright;¹⁴ in 1981 the Supreme Court, in *Diamond v. Diehr*,¹⁵ gave the patent office unequivocal permission to issue patents for certain kinds of computer software;¹⁶ and, at about the same time, with PC-1 the IBM corporation entered the market for desktop personal computers, a market that had been dominated by Apple. Apple had encouraged the rather small coterie of users (often called "hackers") to write their own software and exchange disks in an open and collegial manner. This tradition arose in the academic environment, where computer programmers were supported by educational institutions whose primary function was to produce and share the fruits of research.

Thus when software for the personal computer entered the mass marketplace, the only protection available seemed to be trade secret licenses. As the distributors had no real relationship with those who acquired the floppy disks holding the program, however, the concept of an enforceable contract between the two was tenuous. The lawyers then came up with a questionable solution: The floppies were packaged with a "shrink-wrap" license, which purported to include all the provisos that might have been included in a contract negotiated between more or less equal partners; the user was warned that breaking open the plastic wrapping constituted acceptance of the terms of the contract.

Courts have questioned the validity of the shrink-wrap license, but software developers continue to use it.¹⁷ They also claim copyright protection and file for patents where available. Indeed, in such a fluid legal environment, any prudent lawyer would seek all three forms of protection: trade secrets law, with a shrink-wrap license for mass-marketed software; patents, if the software was truly innovative; and copyright, in the absence of assurance that either of the other alternatives would provide real safeguards.

Software manufacturers continued to seek copyright registration, as the copyright office was hospitable. Thus when the Commission on New

Technological Uses (CONTU) asked a group of copyright lawyers whether the law should be changed to apply copyright law to computer software, there was a warm and positive response, as they already had a large investment in this type of protection. Indeed, one of the arguments put forward for retaining copyright as the legal regime of choice is that copyright lawyers have more skill dealing with computer software than do patent lawyers. Patent lawyers do not agree and continue to pursue patents for their clients quite diligently.¹⁸

Copyright procedures had to change to accommodate the protection of software that was marketed under a claimed protection of trade secret law. Copyright law traditionally covered intellectual work that was fully disclosed through publication (such as books, maps, and magazines) or public display in newsstands and galleries, inviting readers and viewers to see all of the product (such as sculpture and paintings).

Software developers marketed their products with a *source code* (the high-level language used by the programmer) transformed into an *object code* (language understood by the computer) that was incomprehensible to the ordinary user. According to constitutional mandate, copyright and patent protection was not designed to prevent disclosure to the public, but only to ensure a right to compensation in the form of a royalty or patent license fee. But in software design and manufacture, clearly the intent was to impose secrecy about the details of implementation. The copyright office cooperated, by providing that trade secrets could be blocked out in the source code submitted to the office and even, in some cases, that a product could be registered without any identifying material.¹⁹

Lotus and Look-alikes: The Interface Litigation

Once the dust had settled on the decades-long litigation to establish that object code and source code were well within the copyright rubric, the attention of the legal profession turned to the computer interfaces. There are several types of interfaces: (1) those that define the environment through which the user engages the computer's functions, such as menu commands; (2) those that permit software programs to be deployed by computer hardware; and (3) those that permit software programs to interact with one another. In this litigation Lotus has also played a leading role.

Not surprisingly, when look-alike challengers to Lotus 1-2-3 began to appear, Lotus executives and their lawyers were far more aggressive than their predecessors had been in protecting their interests. One of the

early cases against look-alikes involved a competitor that advertised itself as being "designed to work like Lotus 1-2-3 keystroke for keystroke," "a feature-for-feature workalike for 1-2-3"—even advising that "users familiar with 1-2-3 can skip this chapter."²⁰

Lotus sued, claiming copyright infringement. The defendant argued that only source code and object code were entitled to copyright protection, and that the user interface of Lotus 1-2-3 was utilitarian—like the functional layout of gears on a standard automobile transmission, the keys of a piano, or the configuration of controls on a musical instrument—and thus not subject to copyright protection.²¹ The lawyers for Lotus countered that the intellectual effort and creativity required to create a successful interface was more difficult than converting the design to a code the computer could understand. Judge Robert Keeton acknowledged this valuable information asset, following a line of cases that had already determined that nonliteral elements of the computer software—the sequence, structure, and organization—could be protected by copyright.²²

Indeed, according to Judge Keeton, were it not so, and the nonliteral elements of expression "merged" with the idea, leaving only trade secret law for protection, then the length of protection would be very short: "merely the time it takes to examine a program and then duplicate the nonliteral elements in a newly written computer program."²³ In a lengthy opinion, Judge Keeton concluded that the defendants had copied a computer interface—specifically, the menu command structure of Lotus 1-2-3—that was capable of being expressed in many, "if not an unlimited number of ways," and was very much a part of a copyrighted work.²⁴ One fledgling program spawned by the spreadsheet explosion, Microsoft's Excel, was cited in the opinion as a good example of an electronic spreadsheet that avoided infringement by using a different command structure. Judge Keeton rejected the arguments of the defendants that because Lotus's menu command structure was an essential and functional element, thus furthering the utilitarian aspects of the software, it was not qualified for copyright protection:

It may be quite true . . . that things that *merely* utter work, such as the cam of a drill, are not copyrightable. It is not true, however, that every aspect of a user interface that is "useful" is therefore not copyrightable. For example, Lotus 1-2-3 is surely "useful." It does not follow that when an intellectual work achieves the feat of being useful as well as expressive and original, the moment of creative triumph is also a moment of devastating financial loss—because the triumph destroys copyrightability of all expressive elements that would have been protected if only they had not contributed so much to the public interest by helping to make some article useful.²⁵

But there is a strong societal interest not treated in Judge Keeton's argument.²⁶ In the case of the "user interface" in particular, it is clearly in the interest of society for there to be only one system, or very few. Otherwise, every time an organization changes or upgrades its software, it must factor in not only the cost of the new equipment but the often very high cost of retraining its employees to use an entirely new system, adding to the cost of the product.²⁷

Much of the public concern about judicial resolution of these cases is that they militate against industry standardization in the user interface. Unless the similarities are dictated by the restrictions of the particular industry segment to be served (as, for example, cotton growers),²⁸ judges hearing copyright infringement cases have tended to find for the plaintiff (the copyright holder) when there are such substantial similarities program to program that it could be said by the nonexpert to have the same "look and feel."²⁹ If a program could be designed so that the user group could access its features in a variety of ways, having the user press the same keys to bring about the same results has led to an assumption that actual copying did take place.³⁰

However, users generally desire compatibility of programs, especially in a general category of programs. There is nothing more frustrating in network access, for example, than being unable to remember how you "sign off" from the particular program you have been using—that is, how you turn off the meter charging your account. Is the command QUIT, OFF, DISC (disconnect), SO (sign off), \Q (quit), LO (log off), or some other variation? Much damage can be done by having an icon of a container used as "storage" (save) in one program and "garbage" (delete) in another. One frustrated icon basher described the situation as follows:

Graphical user interfaces were ballyhooed as providing an intuitively obvious and consistent interface.

Yet some consistency breaks down when applications use different pictures for the same functions, even basic ones like opening a file or pasting from the clipboard. Maybe software makers feel compelled to use different icons. They do not want to risk infringing the possibly copyrightable look and feel of another package's icons or arrangement of them. Anyway, someone like me who uses a half dozen software packages could wind up mastering 150 or so icons, the hieroglyphics of the 1990's. I'd be as literate as an Egyptian seven-year-old from the time of the pharaohs. (*The Columbia Encyclopedia* informed me that there were only 604 Egyptian hieroglyphs. Most, it said, could be used in up to three different ways.) I fear that a fluent future user may have to master more than six hundred symbols . . . the latter will begin to make our screens look like Japanese or Chinese typewriters.³¹

By encouraging differentiated user interfaces, the law impedes the adoption of widely shared conventions, frustrating users, unnecessarily requiring programmers to reinvent the wheel, pushing up development costs, and inhibiting the compatibility that encourages a competitive marketplace. The challenge is to devise a legal system that encourages standardized user interfaces while rewarding human labor that leads to innovation and progress.

In attempts to accommodate the needs of PC users and software programmers who wished to design compatible programs, various strategies have been devised to standardize user interfaces. Each has its pitfalls. The most obvious was to convince the courts that the expression used in the interface design was so restricted by the use for which it was intended that it "merged" into the unprotectable "idea."³² The second strategy is to convince the court that the similarity is due to a limited number of ways to accomplish the task and that the choice of command is dictated by necessity rather than by a desire to copy a competitor's choice of expression.³³

A third strategy is defensive; companies would seek patents or copyright protection for all of their software in order to be able to cross-license with other companies those products that are needed to serve the marketplace optimally. However, this strategy does not serve small companies well if they do not have a sufficient storehouse of intellectual property rights of their own to offer in exchange at the bargaining table.³⁴ Another strategy, favored by a group of professors of intellectual property law, is to file *amicus curiae* ("friend of the court") briefs in computer software cases, urging a finding of "fair use" for decompiling³⁵ a computer code in order to study and understand the underlying uncopyrightable elements of the software (citing the applicability of section 102[b] of the copyright law, which excludes "any idea, procedure, process, system, method of operation, concept, principle or discovery").³⁶ Further, they argued, since decompilation was the only means of gaining access to the object code, the public would not receive "its fair part of the copyright bargain": the "opportunity to read and be inspired by the work and a free license to copy all of the ideas and processes embodied in it."³⁷

A different strategy is exemplified by the efforts of the European Community (EC) to devise a new form of protection dictated by the needs of the marketplace. After much deliberation, the EC decided to recommend to its members changes in legislation to permit decompilation of computer object code for specified purposes when "indispensable to obtain the information necessary to achieve the interoperability of an independently created computer with other programs."³⁸ The

directive specifically warns against such a decompilation right being used for the "development, production or marketing of a computer program substantially similar in its expression, or any other act which infringes copyright."³⁹ A similar rational and analytical approach has been carried forward in parallel by the U.S. Congress Office of Technology Assessment, with only modest legislative impact to date.⁴⁰

A final strategy is to gamble reaping sufficient rewards from the marketplace to minimize the damage should later litigation not sustain your legal grounds. This was the alternative that Borland International, a competitor to Lotus, must have chosen when it issued Quattro Pro with a similar interface to Lotus 1-2-3, although its own proprietary version of the electronic worksheet was quite different.

When Borland brought out Quattro Pro, with two interfaces, one its so-called native or original interface, and the other an "emulation" interface of Lotus-1-2-3 commands, there was sufficient reason to believe that this simulation might be found to be a copyright infringement. The case was to be heard by the same Massachusetts court where Judge Keeton had made something of a habit of presiding over computer software infringement cases. By then, his decision discussed earlier had been roundly criticized by intellectual property lawyers as ignoring the elements in the software that are excluded by the statute from copyright protection or are already in the public domain.⁴¹ He paid lip service to these exclusions but found sufficient evidence of copying to find Borland guilty of infringing Lotus's copyrights.

By the time Justice Keeton confronted the *Lotus v. Borland* controversy, decisions in other circuits had been rendered criticizing the sweeping precedent enunciated in *Whelan v. Jaslow*, the leading decision in the interface cases. That case involved a software program whose major intellectual contribution was an understanding of the dental profession and its mode of operation. The courts found the user interface designed for the dental profession to be copyrightable.⁴² Later decisions determined that computer software must be broken down into its constituent parts before comparing similarities.⁴³

In the Borland decision (which was settled for half a million dollars and withdrawal of the infringing product from the market),⁴⁴ Judge Keeton modified his sweeping rejection of the Lotus 1-2-3 clone, recognizing the need to filter out unprotected procedures, processes, systems, and methods of operation that are excluded from copyright protection. Nevertheless, he found that Borland had definitely infringed portions of the Lotus 1-2-3 interface but that a jury could find other noninfringing elements. The tests, according to Judge Keeton, were threefold: (1) to filter out the uncopyrightable elements from the copyrightable expres-

sion, for example, the idea, system, process, procedure, or method; (2) to determine whether the "expression" chosen to implement the noncopyrightable elements had been so essential that it could not be distinguished from the nonprotectable elements, and thus could be said to have been merged with one or more of them; and (3) having identified protectable elements of "expression," whether they represent a "substantial part of the allegedly copyrightable 'work.'"⁴⁵

Judge Keeton found that Borland had copied the Lotus 1-2-3 menu commands, menu hierarchy, macro language, and keystroke sequences and that such elements were subject to copyright protection. By its own admission, Borland could have differentiated these commands from those used by Lotus, as its original interface was substantially different from that of Lotus-1-2-3. There were other elements Judge Keeton found to be functional and thus not copyrightable. He found his three tests to be consistent with those enunciated by the Second Circuit in a similar case involving user interfaces, and rejected the sweeping protection offered by the Whelan case while recognizing that the precedent it established was still good law in the Third Circuit until the Supreme Court decided to accept a user interface case.⁴⁶

The Third Circuit follows the Whelan decision in upholding that only the idea behind the program belongs in the public domain and all else is expression, which can be protected by copyright. The Second Circuit rejected this approach in *Computer Associates v. Altai*.⁴⁷ While admitting that the "essentially utilitarian nature of a computer program further complicates the task of distilling its idea from its expression," the court commended the lower court for breaking the software into its component parts, "since each subroutine is itself a program and thus may be said to have its own 'idea.'"⁴⁸ Under this reasoning, it is necessary to sort out which ideas may be replicated while still avoiding replicating the ways in which these ideas are expressed.

The Ninth Circuit has gone even further in freeing the interfaces from copyright protection by determining that disassembly of object code is a fair use of a copyrighted work when used to gain access to unprotected elements, even though the "use triggers a misleading trademark display."⁴⁹ Sega Enterprises, Ltd., a Japanese corporation that developed and marketed video entertainment systems, developed a trademarked security system that required a compatible piece of software to unlock access to its Genesis Console, only to discover that the U.S. courts would permit decompilation of its object code by competitors in order to write games that would run on its console without a licensing agreement.⁵⁰

On the West Coast, Apple Computer was pursuing litigation against Microsoft and Hewlett-Packard for use of what it perceived to be a pro-

prietary interest in its popular windowing software. Although the case was somewhat more complicated than the Lotus cases, because it involved a license agreement between the parties to use some of the Apple software, nonetheless, in what has been touted by some intellectual property lawyers as "finally getting it right" the court held as a matter of law that only 4 of 150 or so elements in the software alleged to be infringed were sufficiently questionable to send to the jury.⁵¹ In addition, after filtering out all of the licensed aspects, those residing in the public domain, those specifically excluded by section 102(b) of the copyright statute, and those dictated by the necessities of the functionality or by having become common practice in the industry, what was left would have to be "virtually identical," not merely "substantially similar," to constitute infringement. A 1985 decision predicting that the ultimate fate of software protection would be quite thin was justified.⁵² The protection provided in the last series of cases (other than the Lotus litigation) seems virtually transparent. Perhaps after a decade of litigation under the software rubric, the emperor wears no clothes.

The Current Curious State of Litigation

Today the various circuits of the appellate system in the United States are in conflict over what portions of software can and cannot be protected by copyright. It is now well established that both the object code and source code can be protected from literal copying. Yet it is still not clear how much or how little of the visible command structure, the "look and feel" of the software, can be replicated by others in order to appeal to the user's preference for the familiar. Some courts are still struggling with perplexing principles that they find unworkable, while others are working hard to understand both the complexities of the law and the underlying economics of software production and marketing.⁵³

No consensus has emerged among either lawyers or software programmers and developers over the optimum level of protection for software or under which legal rubric. It is not surprising that those involved in the original decision of CONTU, recommending the deployment of the copyright law for computer software, have lauded the efforts of the courts to conform the copyright law to the new technology,⁵⁴ commending the judges trying these cases for their increasing "computer literacy."⁵⁵ However, other lawyers find the case law over the last five years "collectively a mess."⁵⁶ Anthony Clapes, a leading proponent of copyright as the law of choice for software,⁵⁷ has roundly criticized the Altai case as a "legal Chernobyl," finding that "the court has tortured the

copyright law into an unrecognizable unusable *sui generis* form of protection for software."⁵⁸

The underlying problem is that the courts, while trying to address one of the major concerns of software programmers and users—achieving a compatible interface—have led the law into a corner where the “expression” left to be protected by the existing copyright law is *de minimis*.

By the time the courts have sorted out and excluded all ideas, processes, systems, procedures, and functionality dictated by the hardware or compatibility with other software, or derived from public domain, there may be little left to be protected under the copyright statute other than the right to decompile the object code. If that is arguably permissible under the doctrine of fair use or excluded from contract restrictions by virtue of preemption under the copyright statute,⁵⁹ then what remains are only those innovations that are so truly revolutionary, new, and nonobvious that they qualify for patent protection or that expression so different from any other that it is incompatible with common usage. This state of affairs may portend the end of an extended period of judicial ferment attempting to tailor existing legal concepts to an elusive and undefinable technology.

As I have noted, many in the software industry criticized Lotus when it first started filing infringement suits. Bricklin, who created VisiCalc, called the litigation “[t]he worst thing to happen to software, ever.”⁶⁰ He found the use of compatible interfaces far too important for the health of the software industry to permit a monopoly over the functionality of popular programs: “We all borrow from the best of others. . . . That’s what progress is all about.”⁶¹

Scott Davis, a consulting engineer at the Digital Equipment Corporation, agreed, seeing an awful lot of waste in program writers trying to create variations on what have already been established as workable components. He preferred a situation in which the best interface came to be established as a standard, with programmers then free to direct their energies forward rather than backward: “What you want to do is build on what somebody else has built and not reinvent what was on the bottom.”⁶²

The concept of each generation standing upon the shoulders of previous generations is a long-standing tradition in the sciences. But scientists compete in a different environment, nurtured in research institutes and university laboratories, where salaries are more or less assured and professional rewards and recognition are meted out through an elaborate system that includes literature citations, research grants, and prizes. Software professionals work in a myriad of small and large software houses that rely for their financial health, if not their very survival, on the reactions of the marketplace to their most recent innovation.

The Special Importance of Computer Software

The legal controversy might have remained of small interest to the public at large, except for the substantial impact that computer software has upon our economy, our politics, and our social lives. Computer software, as essential today as steel production was to early-twentieth-century industrial states, constitutes the crown jewels of an information economy.⁶³ The healthy functioning of an information-based economy relies heavily upon computer software, not only for publishing manuscripts (the traditional province of copyright) but for running computer-aided manufacturing (traditionally the province of patent law), as well as our national transportation system, national defense, education, and a myriad of other national and local enterprises. There is hardly an aspect of modern life that does not depend heavily upon computer software.

A strong computer software industry is a major asset of the U.S. economy. Japan and the European Community, as well as Singapore and Australia, have targeted the software industry as critical for broadly based economic development. The American competitive advantage continues; over the last twenty-five years, the number of U.S. domestic software firms has quadrupled, and income has grown from \$250 million a year to well over \$100 billion in 1990, accounting for more than 40 percent of the global market for mass-marketed software.⁶⁴

Because computer software is essential to most other sectors of the economy, it is considered a "driver technology."⁶⁵ It is software that makes possible computer-aided design of manufactured goods. Robotics, the brightest promise for increased productivity and quality control in manufacturing, can function no better than its software. Computerized communications systems facilitate the transfer of instructions, orders, inventory control, even diagnostics—not only for manufacturing firms but for service institutions as well. Such rapid and sophisticated communications permit us to deal effectively in an increasingly global marketplace. Computer software is critical to the takeoffs and landings of air carriers. News and entertainment are enhanced by complicated computer-assisted graphics. Administrative processes are expedited by computerized records systems; and administrative decisions improved through the use of modeling techniques.

The world economy, as Walter Wriston has observed, now operates on a twenty-four-hour "information standard" that disseminates data continuously to millions of computer terminals worldwide, which are used to make financial and market decisions. These decisions, in turn, through the magic of computers, almost instantly become part of the data stream circling the globe and informing the world economy.⁶⁶

The stock market, long computer-dependent, is now computer-vulnerable as well. The precipitous drop in the stock market in the fall of 1989 was attributed primarily to selling waves dictated by widely used computer-driven analyses that automatically triggered sell orders when certain predetermined price levels were reached.⁶⁷

Many of us have become personally dependent on our computers to help us manage our daily lives. We use WordPerfect, Microsoft Word, Ami Pro, or some other word-processing software for our correspondence. Quicken writes our checks, and Lotus 1-2-3, Excel, Borland's Quattro Pro, or some equally efficient spreadsheet, manages our financial affairs. We keep our calendars on Sidekick or Windows. We use electronic paintbrushes. We relieve our tensions with screen versions of solitaire and mah-jongg, Dungeons and Dragons, or Amazon. We may even have an F-15 fighter-pilot simulation on our hard drive.

Those of us who have attached modems to our personal computers may be exploring the frontiers of cyberspace, an electronic environment populated only by the computer-literate and their console cowboy pals. Those whose computer literacy is not founded in the requisite fanaticism may restrict themselves to hopping around on public networks like CompuServe or Prodigy, but millions of us are transporting messages electronically over the Internet to users far and wide across the globe for professional, political, and social purposes.

It is this ability of computers to speak to each other that makes compatibility of software so important. There are so many different protocols for getting into and out of these various software environments that we may be inhibited by technical barriers or just plain user incomprehension. For this reason, broadly based copyright protection seems ill suited for software. Other types of intellectual assets protected by copyright are works of art and literature, for which exact copying would be offensive, if not illegal, and which would lead to dull exhibitions in museums and repetitious readings in libraries and coffeehouses.⁶⁸ More important to the user of software is the ability of a standardized interface to access a number of different applications, as well as its ability to fit into a larger universe or networked environment. Compatibility with other operating systems, as well as other software applications, is necessary if the user is to deploy the computer as a gateway to communicate with other machines, telecommunications carriers, and software applications. While an exhibition of artists, each showing different techniques and work products, is pleasing, a cacophony of software programs all bleating out a different set of command structures not only boggles the mind but inhibits the user from migrating to different platforms.

On the other hand, allowing copying of interfaces in computer software leads to customer satisfaction and permits small entrepreneurs to

concentrate on building on the platforms provided by major software houses, providing diversity in the marketplace and enhancement of applications for specialized groups of users. Harry Reinstein, former CEO of the Aion Corporation, finds it desirable for all interfaces to be open. Interfaces represent, at least in his view, a competitive point of entry critical to maintaining the competitive environment of the software industry.⁶⁹ It may be that the decompilation of code to understand the underlying functionality can be tolerated in the public interest as a necessary social value if some value has been added, improving the general state of the art. However, such a concept of fair use based upon a redeeming-social-value standard ignores the fact that the programmer who improves upon the state of the art has used valuable information assets that, in an information market, should demand and receive market value.

Those who espouse the concept of "freeware" oppose all types of protection, whether copyright or patent or trade secrets, as placing unnecessary burdens upon programmers, requiring them to go back to the beginning and redesign every system to which they would like to make an addition or add a refinement. They consider the legal requirements as excessively restricting developmental work, which must necessarily be expedited by the freedom to tinker with what has already been done, debug the programs, and alter them to improve their functionality for the particular user.

On the other hand, declaring a desirable interface public domain or requiring compulsory licensing at reasonable rates seems hardly fair to developers who have devoted huge amounts of time, resources, and personal commitment to the design of these now very attractive and desirable information assets. The more resources firms allocate to development, the higher the incentive to maintain the assets as proprietary and to derive income from monopoly control of the product. Thus we have the dilemma of trying to allocate fair compensation while assuring that everyone has access to the same building blocks.

In neither patent nor copyright law is the "sweat of the brow" important. Regardless of the amount of human labor invested, it is only the offer for sale of "copies," the payment of the license fee for use of the "invention," or the execution of a contract for the sharing of a "secret" that triggers legal protection for intellectual property. There should be some legally acceptable way to compensate aggrieved parties whose work product has been misappropriated while respecting the Supreme Court's recent reprimand that the "sweat of the brow" cannot be protected under the copyright statute.⁷⁰

Critics of the freeware approach point out that what actually occurs in the sharing or unauthorized acquisition or piracy of information assets is

the misappropriation and exploitation of intellectual work products without compensation to those who produced it. No one will challenge the right of gifted programmers who produce valuable information assets to share these assets with whomever they like or, indeed, to donate them to the public domain. But it must be noted that most programmers who do so derive income from other sources.

These are quite different circumstances from software programmers writing for the mass market, whose financial survival depends upon the numbers of disks sold or the number of bytes downloaded from an on-line electronic delivery service. There are many alternative devices for compensating the labor expended in producing information assets. However, in an information society the privatization of these assets in the competitive marketplace demands a mechanism for deriving a fair income from the exchange of these assets in that marketplace.

In the past such intellectual creativity has usually been embedded in a product whose very purchase ensured fair compensation to the person who had conceived such a use for it. In an information economy the intellectual productivity or information asset is often marketed and distributed disembodied from the product that will put it to use. Indeed, computer software can be marketed and distributed in electronic impulses that pass quickly and irretrievably over telecommunications lines to distant parts of the globe. Thus misappropriation law or unfair competition may offer a far better theoretical framework within which to analyze and identify miscarriages of justice than "copy," "secret," or "invention." Certainly there is room for innovation in the law as well as in the realm of manufactured products.

There is no guarantee to the creators of intellectual property or information assets that they will benefit from uses subsequent to that of the first purchaser. Case law currently stands at the point of not permitting unconscionable exploitation of the commercial value of another's work product. However, in our global village, even that legal distinction fails to protect innovators from infringing uses of their work products outside the jurisdiction of courts accessible to the innovator.

Are There Alternatives to the Deadlock?

Merely putting the three types of protection (patents, copyright, and trade secrets) together into a single unified administrative framework would have much to recommend it. Entrepreneurs and their lawyers would not have to seek multiple layers of protection in different agencies, different jurisdictions, and attempts to follow the lead of complex,

often indecipherable court decisions. Requiring software innovators to settle on one kind of protection, as some suggest, would not be efficacious. Each system has weaknesses that would leave portions of the software without protection, and each has strengths that if incorporated into a unified system could contribute to a more optimum exploitation of computer software.

On the other hand, an entirely new paradigm might offer a kind of custom-designed protection, with less confusion than reliance upon the existing legal systems designed for the resolution of very different tensions. It might be more appropriate to think in terms of information assets rather than intellectual property. A legal system designed to prevent copying seems an odd model to use in an industry that provides the user with several copying programs as integral parts of every operating system. Indeed, the first thing that software developers recommend their customers do upon opening the package is to make a working copy of the disk, storing the original away in case the copy is damaged or lost. This copying is specifically permitted by the copyright statute.

As a next step we may copy the program to our hard disk. To use the program, we must copy essential parts of it to resident memory. There are programs we can buy that will help us download programs and data from remote data bases, and most programs we use allow easy saving or storing of our work product, in effect copying it from computer memory to a disk, hard or floppy.

In fact, while we can read a book or magazine article without going near a photocopy machine, there is very little that we do with a computer that does not involve copying in one form or another. Thus it would appear to be disingenuous of software sellers to claim shock at the exercise of this basic computer function. It would also appear to be in the interests of an information society to encourage copying rather than discouraging it.

A new paradigm could focus on the uses of information assets rather than the copying of intellectual property. We could avoid using copyright or patents or trade secrets. It would be necessary only to sort out which uses are permitted, which are prohibited, and which require compensation to be made to the originating party. These several categories can be sorted out into the following "uses":

- *Nonuse.* This would preclude literal copying of work product in order to compete with the creator.
- *Authorized Use.* Specifically sanctioned uses, such as for archival purposes referenced in the copyright statute, would be included, as well as all uses licensed by the proprietor of the software.

- *Misuse*. This category would entail all negligent behavior that results in damaging consequences that fairness dictates should be compensable through court action.
- *Abuse*. This category would cover all destructive behavior prohibited by statute and might entail criminal sanctions.
- *Unauthorized But Fair Use*. Such use would likely permit decompilation of code for teaching purposes and for the custom tailoring, for personal or institutional use, of the software by and for the benefit of anyone in lawful possession of the software.

And here are some of the principles that might govern a more coherent legal system for computer software:

1) Ease of registration without lengthy wait for verification that the innovation advances the prior state of the art. This is borrowed from the copyright law. Substantial similarity, as under the present law, would be sufficient to raise a presumption of infringement but would be rebuttable by a showing of independent creation.

2) Full disclosure, which is an underlying principle behind the constitutional origins of both copyright and patent law.

3) Deposit of source code with a receiving office (perhaps the copyright office), where it would be sequestered during the term of protection and released for public access at the end of the statutory period. Such secrecy is borrowed from the trade secrets law but would be available only for a limited period.

4) The statutory period should be long enough to recoup development costs and to provide incentive for innovation as well as reward for "sweat of the brow" that results in utilitarian software. The period should be designed to be short enough that leaving the licensing to the marketplace should not lock or preclude the use of desirable interfaces. A period no shorter than five to seven years and no longer than ten to twelve years would seem appropriate.

5) All actual uses would be subject to compensation during the protected period, except those deemed to be fair personal and noncommercial use or modification of programs to accommodate needs presented in the normal course of business use (using the software for the purpose for which it was intended).

6) Licensing of the uses of the software would be left to the marketplace. If there is sufficient showing of abuse of a dominant position, there are the antitrust laws to fall back upon. Alternatively, some procedure for challenging refusal to license and an appeal to a procedure for compulsory licensing of software interfaces that have become industry standards might be appropriate.⁷¹

Conclusions

Protecting the crown jewels of the information economy is expensive, time-consuming, and frustrating for lawyers, software programmers, managers, venture capitalists, and users. And it is not easy to determine how to treat computer software in an economy whose well-being depends upon a healthy software industry.

The traditional legal framework for balancing interests between public and private claims on intellectual property has served us well as agricultural and industrial societies. But the advent of an information economy has created tensions between these claims and an imbalance between public and private interests. Recently several interrelated factors have eroded the effectiveness of traditional legal regimes for protection of intellectual property: (1) the development of new communications and information technologies, which increase the number of participants in the marketplace, speed the delivery of their intellectual products and services, and facilitate the rendering of perfect copies; (2) the globalization of the marketplace; (3) the independent sale of information assets separate and apart from manufactured products; (4) the trend toward greater privatization of the information marketplace; and (5) the advent of computer software as a hybrid technology that straddles traditionally discrete areas of copyright, trade secret, and patent protection.

An information economy revolves around the compensated use of information assets. Labor is the value added to facts that are gathered and processed into transmittable data. Arguably, if the labor is not useful, it should not be compensated. If it is not only very useful but, in fact, is used, then it should be compensated by the marketplace or by some socially acceptable alternative, such as tenured professorships, Nobel Prizes, or government- or industry-supported research institutes.

Software programmers and their employers in the private sector are concerned that they will not be compensated for the work they do; venture capitalists are concerned that while no one comes along and offers to share their losses when projects fail, there are many who have claims on the fruits of their successes; users are concerned that the copyright law will push the programmers in the direction of making every system so different from every other one that we will have to decide on a lifetime computer system while still in our teens; even the judges are beginning to be concerned about their ability to cope with the fast-changing technology and the challenges in accommodating the law optimally to the needs of the software industry.

The law may currently be adequate, but adequate may not be good enough. Aside from the problems of harmonizing a global infrastructure

for information transfer, it may be no more difficult to modify the present system than trying to persuade the rest of the world to adopt our present overlapping system of trade secrets, copyright, and patents.

Old laws are like hand-me-down clothes. They can be made to make do, but a new suit or dress might fit better, make us feel better, improve our performance. In accord, more than two hundred years ago, was Thomas Jefferson. Inscribed on the marble encircling the memorial to his role in founding the nation is the following:

I am not an advocate for frequent changes in law and constitutions. But laws and institutions must go hand in hand with the progress of the human mind. As that becomes more developed, more enlightened, as new discoveries are made, new truths discovered and manner and opinions change, with the change of circumstances, institutions must advance also to keep pace with the times. We might as well require a man to wear still the coat which fitted him when a boy as civilized society to remain ever under the regimen of their barbarous ancestors.

Has the role of computer software become so critical to the economic health of the nation and the world that it deserves its own system of legal protection, extracting the optimum practices from existing legal regimes?