From: Computers, Ethics & Social Values
D.G. Johnson & H. Nissenbaum (eds)
Prentice Hall: 1995

▷ SHOULD I COPY MY NEIGHBOR'S SOFTWARE?[1]

Helen Nissenbaum

INTRODUCTION

Consider the following situation: Millie Smith is pleased with the way the home bookkeeping application, Quicken, organizes her financial records, even printing checks. Knowing how useful this would be to a good friend of hers, Max Jones, who lives precariously from one paycheck to the next, and yet knowing that the program's price tag puts it outside of Max's financial reach, Millie is tempted to help Max out by offering him a copy of hers. She has read the lease agreement on the outside package which prohibits making copies of the diskette for any purpose other than archival backup, so she suspects she might be breaking the law. However, Millie is not as concerned about breaking the law (nor about the second-order question of the morality of law breaking) as she is about violating moral principles. If she is to copy Quicken for Max would her doing so be justifiable "not so much in a court of law as in the court of conscience"?[2] For private consumers of commercial software Millie's situation is all too familiar.

Although the majority of these private end-users admit to frequently making and sharing unauthorized copies, they experience a nagging and unresolved sense of wrong-doing. Posing as the "conscience" of these wayward software copiers, a vocal group, whom I refer to as supporters of a "strong no-copy view," urges users like Millie Smith to refrain from unauthorized copying[3] saying that it is always wrong. Jon Barwise, for example, in promoting a strong no-copy position, concludes in a series of scenarios whose protagonists must decide whether or not to copy an $800 piece of software, that even in the case of a professor providing a copy of his diskette to a student who needs it to finish a dissertation, "we should answer all of the (above) questions no."[4] Green and Gilbert, in an article directed specifically to users in educational institutions, recommend that "campuses should view and treat illegal copying as a form of plagiarism or theft" and that they should pursue ways of reducing "illegal and unethical copying."[5]

In the following discussion I challenge the no-copy position, arguing that it emphasizes the moral claims and interests of software producers while failing to consider other morally relevant claims—most notably, those of the private end-user. Accordingly, Millie would not be violating moral principles if she were to share a copy with Max. I show that there are morally compelling factors that motivate many acts of software copying, not simply brazen self-interest, irrationality, or weakness of the will. Although I argue that in *some* cases copying is not a violation, I do not support the position on the other end of the ideological spectrum, which completely rejects the constraints of software copy protection. Rather, we need to judge distinct types of situations according to their individual merits. In some situations there will be an overriding case in favor of copying, in others not. In still others, agents confront a genuine dilemma, trying to respond to equally convincing sets of opposing claims.

To reach this conclusion I focused on the arguments, both consequentialist and rights based, that have been proffered in support of the strong no-copy position.

Upon analysis I find that, as a universal position, a strong no-copy position is not defensible.

Two Caveats

First, a word on how I set about recreating the justifications for a strong no-copy position. I've drawn from pieces written for computing trade publications, other non-philosophical journals, electronic-mail communications, as well as conversations. Although the arguments given in favor of a moral prohibition on copying are generally not presented here in a framework of traditional ethical theory, I find this framework useful in organizing and evaluating them. For example, I classify the arguments that predict undesirable consequences of unauthorized copying under the general heading "consequentialist arguments." In a second working category, I classify arguments that claim unauthorized copying to be violations of moral rights and respect for persons. Although this group is more of a grab bag, the label "deontological/rights-based" captures its hybrid spirit. My first caveat, however, is that while the philosophical categories are enlightening, suggestive of potential strengths and weaknesses of the arguments, they should be viewed as rough guides only. Moreover, because few of the commentators offer explicit or complete treatments, I've taken liberties in filling in steps. While I fleshed out the arguments and filled in gaps, I tried to stay strictly within the parameters set by their originators.

Second, in order to simplify the discussion I assume throughout this discussion that programs are written and owned by a single programmer. In the real world of commercial software, teams of software developers rather than single programmers create software products. And for many products, the title usually goes to the software corporation, rather than directly to its employees, the program's authors. In other instances, it goes to intermediate agents such as marketing firms, or vendors. The assumption of a single programmer, should not affect the substantive moral thesis.

Consequentialist Arguments

According to the arguments in this category, it is morally wrong to make unauthorized copies because doing so would have negative consequences. Although copying might appear to offer a short-term gain for the copier, the longer term and broader ramifications will be a loss for both consumers and producers alike. Barwise, for example, charges, "... software copying is a very serious problem. It is discouraging the creation of courseware and other software, and is causing artificially high prices for what software that does appear."[6]

Barwise's remarks suggest that we can expect at least two types of negative consequences. The first is a probable decline in software production. Because copying reduces the volume of software sales it deprives programmers of income. With an erosion of potential revenues, fewer individuals will be attracted into software production. A smaller population of programmers and other software personnel will result in a reduction of available software. Furthermore, a slowing in software development would have a dampening effect on general welfare. The second negative impact of copying is a projected rise in software prices. Wishing to recoup anticipated losses caused by unauthorized copying, programmers will charge high prices for their software. Giving as an example Wolfram's *Mathematica*, which in 1989 was

priced at $795, Barwise blames copying for the artificially high prices of software applications. How good are these arguments?

Embodied in the consequentialist line of arguments are a number of empirical assumptions and predictions which, I contend, are open to challenge. For consequentialist arguments to provide a moral as well as a prudential rationale, they must demonstrate links between copying and reduced income, between reduced income and decline in the software industry, and decline in production and an overall decline in society's welfare. If copying hurts the software industry but has no effect on general welfare a prohibition is not morally justifiable on consequentialist grounds. If copying is not directly related to income, nor income to a decline in the industry, then too, the argument breaks down. On close scrutiny these links don't stick. Furthermore, even if some damage could be attributed to unauthorized copying, I conclude that it's insufficient to warrant the all-out prohibition of the strong no-copy position.

Consider the claim that unauthorized copying leads to loss of sales. Although on the face of it, the argument is compelling, the implied link between copying and reduced sales is not always direct. Imagine a situation in which you are deciding whether to buy software application A or copy it from a friend. Although the consequentialists would have us think of all instances of copying as situations in which an agent must decide between the exclusive alternatives, buy A, or copy A, in many real-life situations this is not so. Computer users copy software that they would not buy for a number of reasons: because they could not afford it; are not yet sure that they want the product; or quite simply, have placed higher priority on other needs. For them, the choice is: copy A, or not have A.

Moreover, copying can actually lead to an increase in overall spending on computer software, at least for some individuals. Software sharing opens opportunities for trial and experimentation to otherwise timid users who thereby grow more comfortable with computers and software. As a result they become more active and diversified consumers of software than they would have been without those opportunities. We also find that users who are impressed by a particular piece of copied software, in order to own the manual and enjoy some of the additional benefits of "registered users," will go on to buy the application. In other words, much unauthorized copying would not result in loss of sales and some, in fact, would lead to increases.

The prediction that reduced income will discourage further creation of software belies a complicated story about motivation, action, and reward. Whereas wholesale fluctuations and extreme reductions probably would discourage would-be programmers, the effects of smaller fluctuations are not clear. Richard Stallman[7] ably makes the point that directly tying software production to monetary reward paints an overly simplistic picture of the rewards that motivate programmers. Well-known for his active support of an open environment for information technology, Stallman suggests that besides the satisfaction of contributing to a social good, the fascination with programming itself will keep many of the most talented programmers working. He also raises the question of how much is enough. Although we would not expect many good programmers to have a monk-like devotion to programming and can agree that people work better when rewarded, it's not clear that any increment in reward will make them work proportionately better. (Furthermore, as suggested earlier, we still do not have a realistic idea of the extent to which cases like Millie Smith's actually affects potential earnings.)

Turning the tables on the usual consequentialist chain of reasoning, Stallman counters that prohibitions on copying, and other restrictions on the free distribution of computer code, has the opposite effect on computer technology. It is slowing

progress rather than encouraging it. He and others suggest that the free exchange of ideas and code characteristic of the early days of systems and software development was responsible for the remarkable pace of progress, whereas limiting free exchange would dampen innovation and progress, moreover, laws restricting access to software would favor large, powerful and generally more conservative software producers. With a greater capacity to exert legal clout, they could control the production, development, and distribution of software, gradually squeezing out of the commercial arena the independent-minded, creative software-engineer, or "hacker". Even if we see a proliferation of commercially available software, we may also see a slowing of the cutting edge. If Stallman's predictions are sound, they offer moral justification for promoting free copying of software, and not the reverse.

So far, I have questioned the empirical basis for the claims that link copying with loss of revenue; claims that link loss of revenue with a decrease in software production; and, more generally, claims that link copying with a loss to the software industry as a whole. What about effects on general welfare? At this level of generality it is probably impossible to draw a meaningful connection between software and welfare. To the extent that software is a social good, it is surely through high-quality, well-directed software and not sheer quantity.[8] To discourage a potential copier, an extreme no-copy position must show the clear social benefits of abstaining without which there is little to offset the immediate loss. This question deserves more thorough exploration than I'm able to give it here because the connection between software production and overall utility or welfare, is complex. It does suggest, however, that the effects on general welfare of a particular act of copying would vary according not only to the context of copying, but also to the type of software being copied. It would also need to be measured against the projected utility to the potential copier

Let us now consider the alleged connection between copying and cost and the claim that producers are forced to charge high prices in anticipation of losses through copying. An obvious rejoinder to software producers, like Wolfram, is that if software applications were more reasonably priced, consumers would be less tempted to copy. If products were appropriately priced, the marginal utility of buying over copying would increase. This pattern holds true in the case of recorded music which could provide a model for computer software.[9] Because the cost of a tape, for example, fits many budgets, it is more convenient to buy the tape than search for someone who might have it. Though both the claim and rejoinder appear to hold genuine insights, they leave us in an uncomfortable standoff. Looking at high prices, pointing at consumers, critics say: "It's your fault for copying." Whereas consumers point back claiming: "It's your fault for charging such high prices." The average user apparently cannot afford to buy software at the current rates, and the programmer cannot afford to drop his or her price. Though we may agree that this is not a desirable equilibrium, it's not easy to see who should take the first step out of this circle of accusations. Resolving the standoff requires asking difficult questions about burden. Upon whom do we place the burden of maintaining a healthy software industry—consumer or producer? This question brings me to my concluding comments.

I agree with defenders of a consequentialist line that a prolific software industry with a high-quality output, which provides genuine choices to a wide variety of consumers, is a goal worth striving for. I disagree, however, that prohibiting copying is the only, or best, way of ensuring this. First, I have tried to show that the empirical grounds upon which they support their claims are open to dispute. Moreover, if a consequentialist approach is to be at all useful in guiding decisions about unauthorized copying, then it must distinguish among different types of copying—for their

consequences surely differ. For example, cases like Millie's sharing a copy with Max would have a vastly different effect than cases in which a user places a copy on the software on a public network. Consequentialist moral injunctions should recognize these differences.

Finally, the no-copy position unreasonably focuses on private end-users, placing on their shoulders the onus of maintaining the health of the software market. But consumer copying is but one variable, among many, that affect the software industry. Holding fixed the other variables might serve some interests, but it gives disproportionate weight to the effects of copying. Decisions by commercial hardware manufacturers and even government agencies can significantly impact software. For example, if a hardware manufacturer perceives that a particular software product is critical to the sale of its machines it may, quite rationally, decide to support the software.[10] In addition, software companies have the capability to influence the actions of potential users by offering not only a good product as code on a diskette, but by also including attractive services such as consulting, good documentation, and software updates. In this way they make it worthwhile for the user to buy software, rather than copy it. The many flourishing software companies stand as evidence that good products and marketing works, despite alleged copying. Because other players—namely, government, hardware producers, software companies—have the power to significantly affect the software industry, we should not ignore their responsibilities when we assess the burden of maintaining the strength of software production. It is wrong for the private consumer to be unfairly burdened with responsibility.

Deontological/Rights-Based Arguments

In urging individual consumers not to make unauthorized copies some supporters refer to the "rights of programmers" and "respect for their labor" Regardless of its effects on the general welfare, or on the software industry, copying software without permission is immoral because it constitutes a violation of a moral right, a neglect of moral obligations. Depriving a programmer of earnings is wrong not only because of its undesirable ramifications, but because it is unjust and unfair. And even if programmers' earnings are not appreciably affected by copying, we have an obligation to respect their desire that we not make unauthorized copies. The obligation is absolute, not broken merely at the discretion of the private end-user.[11] Millie ought not make a copy of Quicken for Max because doing so would be unfair, it would violate the programmers' rights. But what are the rights to which these commentators refer; and does all copying, in fact, violate them?

Rights-based justifications of no-copy require a satisfactory resolution to both questions. They not only must identify the rights of programmers relevant to the question of unauthorized copying, but must demonstrate that copying always violates these rights. Supporters usually cite property rights as relevant to the question of copying. A justification of the position should, accordingly, ask whether programmers do in fact qualify as owners of their programs so that they would have the appropriate rights of private property over them. But justification does not stop here. For even if we resolve that programmers do own their programs, it doesn't follow necessarily that all copying will violate their property rights. Or to put it another way, it is not obvious that property rights over programs include the right to restrict copying to the extent desired. A justification of the no-copy position needs a second step, to follow the finding that programmers own their programs. And that is, to show that copying

violates these property rights. Many commentators fail to recognize the need for the second step, simply concluding that owning implies an unlimited right to restrict copying.

In the discussion that follows I will spell out the two steps in a rights-based position beginning with the question of private ownership, and then moving to the question of whether owning a program implies the right to place absolute limits on reproduction. I will conclude that the second step is the weak one. As before, in recreating the arguments I've worked from informal written pieces, electronic mail messages, and verbal communications. In some cases this has meant filling in missing steps; steps that I judge necessary to making the best possible case for a rights-based justification. Finally, though recognizing that some might object to the very fabric of rights-based justifications of moral injunctions, I offer my criticisms from within this framework, and will not challenge the very idea of a rights-based approach.

Programming and Private Property

First, let's examine the following claim: Because a programmer writes, or creates, software he or she owns it. For some, this claim is so obvious as to not even need justification. To them, a program is an extension of the person's self and so, obviously, belongs to that person. For others, labor theories of property such as John Locke's, which claims that when individuals invest labor in a previously unowned item they earn property rights over it, offer a more traditional moral grounding for private ownership over programs. Locke writes, "Thus Labour, in the beginning, gave a right of property, wherever any one was pleased to employ it upon what was common."[12] Because programmers invest labor in creating a program, they are entitled to the "fruits of their labor." Although Locke's theory addresses the somewhat different issue of private acquisition of physical property, such as parcels of land and harvests, and focuses on the taking of initial title over a previously unowned item (or one held in common), his theory adapts well to intellectual labor. In fact, the case of intellectual property is somewhat easier for a labor theory in that it avoids a common pitfall identified by Locke's critics who, in the context of physical items, worry about the morally "correct" mix of labor with the physical entity.[13] I will concede then, that a programmer, in producing a program, accrues property rights over it, accepting as justification for this claim—if it is even needed—basic ideas of a labor theory of property.

Some have questioned the justice of extensive property rights over programs claiming that software creation is an essentially cumulative activity. Most programs, draw heavily on work that has preceded them so that giving rights to the programmer who happened to write the line of code in question rests on the unwarranted assumption that we can tell accurately where one programmer's labor really begins and the other's ends. For example, most commercial software on today's market is the product of a long line of cumulative work most notably Lotus 1–2–3.[14] However, this objection does not challenge, rather it implicitly adopts, a form of the labor theory because it suggests that *all* those who contributed their effort toward creating a software product deserve proprietary rights over it, and not just those who happen to cross some arbitrary finishing line first. Just because they have made a bigger marketing effort, happen to be more worldly, belong to a large organization, or have good legal representation, does not vest in them a stronger moral claim. Although the question of just rewards for joint labor is an important one in light of the history of the development of computer software, for the remainder of my discussion, I will assume that we can talk meaningfully about *the* programmer who contributed most significantly to a program's creation. It is about this programmer that the discussion about property rights that follows applies.

As stated earlier, showing that programmers own their programs is not sufficient for a no-copy position. Its supporters must still demonstrate that owning a piece of software implies a moral right to restrict copying to the extent desired (and thus the duty in others to refrain from copying). How might I demonstrate this "second step," required of a rights-based justification of strong no-copy? In the next section I will examine whether a universal prohibition on copying software necessarily follows from general property rights over it.

Owning Software and Prohibiting Copying

In general, ownership implies a set of rights, rights defining the relationship between an owner and a piece of property. Typically the rights of an owner over private property fall into a number of set categories including: one that covers conditions on initial acquisition over a previously unowned object;[15] another that refers to the extent of use and enjoyment an owner may exercise over that property; a third that determines the extent to which an owner may restrict access to her property (or alienate others from her property), and a fourth that endows upon an owner the power to determine the terms of transfer of title. Thus abstractly conceived, the concept of private ownership yields a fairly well-defined set of rights. When instantiating these rights in actual cases of owning a specific given item, the specific rights an owner has over that item, can vary considerably according to a host of factors. First, at the most general level, certain social, economic, political, and cultural factors greatly affect our ideas about private property rights, their nature and extent, and what sorts of objects can be owned privately in the first place. To simplify matters, for purposes of this discussion, I will assume a common background of roughly Western, free-market, principles. A second variable that also significantly determines the specific rights an owner can have over an item[16] is its metaphysical character, or type. For example, the specific rights a child has over his peanut butter sandwich might include the rights to consume it, to chop it into twenty pieces and to decide whether to share it or not with a friend. But such rights make no sense in the case of landowners and plots of land, pet owners and their pets, car owners and their vehicles, and so forth. When we determine the appropriate set of rights instantiating the general rights of use and enjoyment, restricting access, terms of sale, on items of varying metaphysical character, we come up with distinct sets of specific rights. Whereas intellectual property stretches classical ideas of locking away or fencing ("restricting access"), consuming ("use and enjoyment"), and bartering ("transfer of title") deciding what it means to own software poses an even harder puzzle.

Computer software has raised a host of challenges to property theory, testing the traditional concepts and rationales in novel ways. Because even relatively simple programs have numerous components and moreover have various aspects, the first problem is to define, or identify, the "thing" that is *the* program, the thing that is the proper subject of private ownership. A program can be identified by its source code and object code, a formal specification defining what the program does, its underlying algorithm, and its user interface, or "look and feel." Each of the various compo-

nents—or aspects—has a distinct metaphysical character and consequently suggests a distinct set of property rights. For example, because a program's source code is considered similar to a written work, it it considered by most to be covered by copyright laws. By contrast some judge a program's algorithm to be a process (and not a mathematical formula) and thus claim that it is patentable. Legal debates address the issue of whether one can abstract a program's so-called look and feel and claim to own that, in addition to, and independently of, the code, algorithm, and so on. And if so, they argue over whether legal protection ought to be through copyright, patent, or something else. There are many instructive works dedicated to the question of the optimal form of legal protection of all these aspects of software in a growing literature which is written from legal, philosophical, and technical perspective.[17]

Fortunately, we need not wait for a resolution to the entire range of puzzles that software ownership raises in order to gain a better understanding of Millie's dilemma. We acknowledge, in her case, that she explicitly duplicated object code, and thus we bypass many of the complexities. However, it is important to note the existing backdrop of uncertainty over how to categorize the metaphysics of software, and thus, how to fit it into our network of ideas on property rights. We are drawing conclusions about software ownership on the basis of imperfect analogies to other forms of private property. This leaves open the possibility of significant differences.

We are ready now to return to this section's central question of how one might derive a prohibition on copying from ownership. On the basis of earlier observations about private property we can conclude that a programmer, or owner, has rights over the program including rights to restrict access and rights of use and enjoyment. Presumably, the programmer's right to generate earnings from his program would instantiate the latter. The programmer could choose to give others limited access to her program by selling diskettes, upon which she has copied the program, at a price she determines. But because the programmer still owns the program itself, she may impose restrictions on its use—in particular she retains the right to prevent buyers of the diskettes from making copies of the program that she has not explicitly authorized. Thus, we derive the programmer's specific right to restrict unauthorized copying from the general right property owners have to restrict access by others to their property. To distinguish transactions of this type from other types of sales, commercial software vendors adopt the jargon "software license" rather than "software sale." Thus, the argument from rights would dictate that Millie not copy because doing so would violate a programmers' valid claim to both use and enjoy his or her property (by depriving them earnings) and restrict access by others to it (by making unauthorized copies).

But this picture leaves out an important component of property theory. Like other rights, property rights restrict the freedoms of others by imposing certain obligations on them. For example a promisee's rights imply an obligation on the part of the promiser to keep the promise; a landowner's rights implies an obligation on would-be trespassers not to cross his or her land. As I stated earlier, the precise nature of property restrictions will vary according to the metaphysical character of the property. But there is yet another factor that shapes the extent and nature of property—and in fact all—rights. Even theorists of a libertarian bent, who support extensive rights over private property, recognize that these rights are not absolute. For example, Locke argued that morality allowed the appropriation of previously unowned property only "where there is enough, and as good, left in common for others."[18] And Nozick, also recognizing limitations on property rights, illustrates one source of these restrictions with his colorful example: "My property rights in my knife allow me to leave it where I will, but not in your chest."[19] In other words, although owning a knife implies extensive rights of use and enjoyment, these rights are constrained by justified claims, or rights, of others—in this case, their right not to be harmed. While I wish to avoid either endorsing or criticizing the more far-reaching agendas of these two authors, I want to draw attention to an important insight they offer about private property rights: that property rights are subject to the limitations of countervailing claims of others.

Actual practice demonstrates that, as a rule rather than an exception, when we determine the nature and extent of property rights, we acknowledge the justified claims of others. For example, in determining the rights of the owner of a lethal weapon we're influenced not only by its general metaphysical features (when we determine the types of actions that constitute use and enjoyment), but are concerned about the well-being of others. And so we restrict the way people may carry lethal weapons—either concealed or unconcealed depending on the accepted wisdom of the city or state in which they happen to live. We regulate construction projects of urban property owners for far less concrete counterclaims than freedom from bodily harm, but in the interest of values like aesthetic integrity of a neighborhood, effects on the quality of life of immediate neighbors, and so forth. We restrict the rights of landowners over water traversing their land, preventing them, for example, from damming a flowing river. We also constrain the behavior of motor vehicle drivers. In all these cases where we perceive a threat to justified claims of other individuals, or of a social order, we limit the extent of owners' rights over their property. It makes sense to carry this principle over to the case of software asking not only about the claims of programmers, but the claims of end-users.

Does Millie Smith have a reasonable counter-claim that might limit the extent to which Quicken's owners can constrain her actions. She would like to duplicate her Quicken software for Max, an act of generosity, helping satisfy a friend's need. Despite the programmer's preferring that Millie not share a copy, Millie is motivated by other values. She views making a copy as a generous act which would help a friend in need. Copying software is a routine part of computer use. Millie's proposed action is limited; she has no intention of making multiple copies and going into competition with the programmer, she wouldn't dream of plagiarizing the software or passing it off as a product of her own creation. The entire transaction takes place within the private domain of friends and family. She would view offering a copy to Max as a simple act of kindness, neither heroic nor extraordinary. Interfering with the normal flow of behavior, especially as pursued in the private realm, would constitute unreasonable restriction of an agent's liberty. Thus, Millie's countervailing claim is the freedom to pursue the virtue of generosity within the private circle of friends and family.

The conclusion of this line of reasoning is *not* that, from a perspective of rights, *all* unauthorized duplication of software is morally permissible. I am suggesting merely that we decide the question whether to share or not to share in a case by case fashion. Although in some cases a programmer's desire that the user not copy software is a defensible instantiation of the right to restrict access to private property, in others the restriction will not be defensible because it conflicts with the valid claims of another agent. And even in the cases where making a copy would not be immoral it would not follow that the programmer has somehow lost all the property rights over his or her program. Commentators like Green and Gilbert are right to draw attention to programmers' claims over their software, and to encourage respect for intellectual labor; but they overlook the possibility of relevant, conflicting, counterclaims. When,

at the beginning of the paper, I referred to the copier's dilemma, it was the dilemma created by conflicting obligations: on the one hand an obligation to respect a programmer's property rights, which in some cases includes the right to restrict copying; and on the other an obligation to help others, tempered by the belief that one ought not have one's behavior unduly restricted within the private domain.

Consider some objections. One objection is that no matter what Millie might think about helping Max, you just cannot get away from the fact that she's violating the programmer's property rights. And this is the reason that her copying—and all unauthorized copying—is immoral.

This objection fails to recognize that counterclaims can substantively affect what counts as a moral (property) right, in any given situation. Consider the rights of a landlord with respect to a leased apartment. When that apartment is vacant, the owner may come and go as he wishes; he may renovate it, choose to rent it, or to let it stand empty. However, once the apartment is leased, the landlord's rights of entry are limited by a tenant's competing right to privacy. Even if it would suit a landlord to stipulate in his lease the right to make surprise checks, this wish would be overridden by the justified claims of his tenants not to be disturbed, not to have their privacy violated. We would not say that the landlord's property rights are violated by the tenant; we would say that the landlord no longer has the right of free entry into his leased property. Consider another example. Let's say someone buys a word-processing package. On the outside of the customary sealed envelope containing the diskette, the buyer finds not only the usual terms of a lease agreement, but one further condition. The programmers stipulate that consumers are free to use the word processor any way they want, except to produce a document that promotes abortion. They reason that the abortion stipulation is merely an additional instantiation of their rights as owners to restrict access by others to their property. However, I think that the buyer could quite reasonably object that despite the programmer's intellectual property rights over the word processor, these rights do not include the right to control its use to the extent that it overrides valid, competing, claims to freedom of expression. Similarly, Millie, judging that in the private domain she should be largely unrestricted, could argue that the moral arm of the programmer does not extend into the private domain. We conclude, therefore, that in copying for Max she does not violate a moral right.

In a second objection, a critic could charge that if we judge Millie Smith's copying to be morally permissible, this would open the door to a total disregard for the rights of programmers. There would be no stopping agents from making multiple, unauthorized, copies and selling them in competition with the original programmer.

This objection doesn't hold because Millie's case, being significantly different from those other cases, would not lead us down a slippery slope. A potential copier must show a justifiable claim that conflicts with the programmer's. In the objector's example, and even in the case of a do-gooder who decides to place a piece of privately owned software on a public domain network,[20] copying takes place in a public domain lacking Millie's personal and private motivations. They lack the compelling counterclaim. Specifically for the public, commercial arena, we would expect to generate a network of laws and regulations to cover the many cases which moral principles alone could not decide.

Another objection asserts that Millie would be acting immorally in making a copy for Max because copying is stealing. But this objection begs the question because it *assumes* that copying is stealing. In this section we've been examining whether or not copying always violates property rights and therefore constitutes wrongful seizure of another's possession. In other words, whether copying is stealing. This objection assumes that we've satisfactorily established that copying is theft, and thus assumes the issue we're trying to establish.

Conclusion

There is a prevailing presumption—in my opinion a disturbing one—that were we to follow the dictates of moral conscience, we would cease completely to make unauthorized copies of software. Yet when we examine the arguments given in support of that presumption we find that they fall short of their universal scope. The soundness of a rights-based rationale depends on successfully showing that owning software entails a right to restrict copying. I have argued that this step is not obvious, and that at least in some well-defined cases the entailment fails—notably, cases in which there are strong counterclaims. In practice this means that we should give equal consideration to the rights of end-users as well as to those of programmers. To simply insist that property rights override end-user freedoms is to beg the issue at hand.

Consequentialist rationales are also equivocal in that they rest on a number of sweeping empirical assumptions—many of which exaggerate the effects of copying, some of which are open to doubt. Moreover, it places squarely on the shoulders of private end-users the onus of maintaining a flourishing industry when in fact there are other agents well placed to share the burden. Many software manufacturers who have been vocal in their complaints, despite current levels of copying, appear to be enjoying overwhelming successes. Perhaps because they offer incentives like good consulting services, free upgrades, and reasonable prices they raise the marginal utility of buying over copying.

Finding that there are insufficiently strong moral grounds for universally prohibiting copying, I conclude not that all unauthorized copying is morally acceptable, but that that some copying is acceptable. There is sufficient variability in the types of situations in which software users copy to suggest that we ought to evaluate them case-by-case. In cases like Millie's and Max's, the argument against copying is not a compelling one.

Finally, some critics insist that the best approach to solving this issue is a hard-line economic one. Clearly, a rights-based approach, which unearths the usual set of conflicting rights is not helpful and leads us to a deadlock. Let the free market decide. We ought to allow software producers to place any conditions whatever on the sale of their software, and in particular, any limits on duplication. Consumers will soon make their preferences known. Defenders of no-copy say that current commercial software conditions are more or less in that position today, except that users are not keeping up their end of the bargain when they make copies of software. But even from this hard-line economic standpoint, a no-copy line is disturbing because it lets the robustness of a market depend on a mode of behavior to which most do not conform, and many find distasteful, that is, restricting the inclination to private acts of beneficence and generosity. Unless we alter human nature, experience suggests that this would be a shaky equilibrium.

On a final idealistic note, I echo strains of Richard Stallman in observing that if we can eradicate copying only when individuals ignore a natural tendency to respond to the needs of those close to them, we may not be maximizing expected utility after all.

NOTES

1. An earlier version of this paper was presented at the Fifth Annual Computers and Philosophy Conference, Stanford University, August 8–11, 1990. Several members of the audience, with their sharp criticisms and suggestions, helped clarify my thinking a great deal. I'd also like to thank members of Partha Dasgupta's Applied Ethics Seminar at Stanford (1989) for useful and creative comments.
2. David Lyons, "The New Indian Claims and Original Rights to Land" in *Reading Nozick: Essays on Anarchy, State and Utopia*, (Ed) Jeffrey Paul, Rowman and Littlefield, Totowa, New Jersey. 1981.
3. I will not be dealing with unlikely cases in which copy in software might save a life or avert a war. I assume that even those committed to a no-copy position would find rationale to permit those acts.
4. Jon Barwise, "Computers and Mathematics: Editorial Notes." in *Notices of the A.M.S.*
5. K. Green and S.W. Gilbert, "Software Piracy: Its Cost and Consequences" in *Change*, pp. 47–49. January/February 1987.
6. Jon Barwise, "Computers and Mathematics: Editorial Notes." in *Notices of the A.M.S.*, 1989.
7. R. Stallman, "The GNU Manifesto" in *GNU Emacs Manual*. Copyright 1987 Richard Stallman.
8. Joseph Weizenbaum in Chapter 1 of *Computer Power and Human Reason*. San Francisco, Freeman, 1976, makes suggestive comments arguing that consumerism needn't necessarily lead to greater choices among genuinely distinct products. A conservative market might remain unimaginatively "safe," coming up with only trivially diverse products.
9. Although some claim that the loss in sound-quality is a major reason for recorded music being less frequently copied, this doesn't tell the story for all (the average) listeners.
10. Both Stallman, *ibid.* and Barwise, *ibid.* (and probably others) have made similar points.
11. Though strictly speaking, a rule-based approach could ultimately be grounded in utilitarian terms, the ones I consider here merge the rights-based and deontological styles of moral reasoning. They cite programmers' rights, inferring from them absolute obligations on the parts of software users.
12. John Locke, Section 45 in *Second Treatise of Government*, originally published 1690, Hackett Publishing Company, Indianapolis, 1980.
13. Nozick's discusses this problem quite extensively in *Anarchy, State, and Utopia*, Basic Books, Inc. 1974.
14. For an interesting history of software inter-dependence see Bill Machrone's, "The Look-and-Feel Issue: The Evolution of Innovation" in *Computers, Ethics, & Society*, M.D. Ermann, M. B. Williams, C. Gutierrez, Oxford University Press, New York, 1990.
15. This was Locke's central preoccupation.
16. Metaphysical character can co-vary with cultural–social factors to make for an even more complex picture. Consider the potentially diverse views of descendents of European traditions and those of Native American traditions on property rights over land, sea, and air.
17. See, for example: M. Gemignani, "The Regulation of Software," *Abacus*, vol. 5, no. 1, Fall 1987, pp. 57–59; D.G. Johnson, "Should Computer Programs Be Owned?", *Metaphilosophy*, vol. 16, no. 4, October 1985, pp. 276–288, P. Samuelson, "Why the Look and Feel of Software User Interfaces Should Not Be Protected by Copyright Law," *Communications of the ACM*, vol. 32, no. 5, May 1989, pp. 563–572.
18. Locke, *ibid*, Chapter 5 Section 27.
19. *Anarchy, State and Utopia* by Robert Nozick p. 171.
20. I confess to being stymied by cases such as that of a school teacher in a poor ghetto school deciding to make unauthorized copies of a software applications that he believes would help his students, who would not ordinarily be able to afford it.

▷ THE SOFTWARE PATENT CRISIS

Brian Kahin

An explosion of patents on software processes may radically change the programming industry—and our concept of human expression in the computer age.

Last August, Refac International, Ltd., sued six major spreadsheet publishers, including Lotus, Microsoft, and Ashton-Tate, claiming they had infringed on U.S. Patent No. 4,398,249. The patent deals with a technique called "natural order recalc," a common feature of spreadsheet programs that allows a change in one calculation to reverberate throughout a document. Refac itself does not have a spreadsheet program and is not even in the software industry. Its business is acquiring, licensing, and litigating patents.

Within the last few years, software developers have been surprised to learn that hundreds, even thousands, of patents have been awarded for programming processes ranging from sequences of machine instructions to features of the user interface. Many of the patents cover processes that seem conventional or obvious, and developers now fear that any of the thousands of individual processes in their programs may be subject to patent-infringement claims.

The Refac suit demonstrates the vulnerability of the industry to such claims. Patent no. 4,398,249 was applied for in 1970, granted in 1983, and only recently acquired by Refac. In the meantime, software developers have been busily creating spreadsheets and other new products unmindful of patents. The industry accepted copyright and trade secret as adequate protection for its products, and most programmers assumed that patents were not generally available for software.

Never before has an industry in which copyright was widely established suddenly been subjected to patenting. As it is, only a few companies that create microcomputer software have the resources to try to defend against patent infringement claims. Most small firms will be forced to pay license fees rather than contest the claims, even though many software patents may not stand up in court.

In the long run, the costs of doing business in a patent environment will radically restructure the industry. Many small companies will fold under the costs of licensing, avoiding patent infringement, and pursuing patents defensively. The individual software entrepreneur and inventor may all but disappear. There will be fewer publishers and fewer products, and the price of software will rise to reflect the costs.