Indiana University
Multi-campus Modular
Computing Curriculum

Introduction to Computing

Module # 11: Intellectual Property Rights for Computer Programs and Data

Prerequisite Skills

- Understand most or all of the terms word processing, spreadsheet, program, algorithm, menu, display, window.

Goals

- Develop ethical sensitivity to the problem of software copying.
- Understand the legal issues involved in copyright of user interfaces and software patents.

1 A Dialog

(Norm and Holly are at Holly's house, where Holly has just been showing Norm her new word processing program.)

Norm: So you're really in love with the SuperSoft Millenium word processor, Holly?

Holly: Oh, absolutely, Norm! It's just wonderful!

Norm: I think it could be useful to me, too. What did it cost?

Holly: Four hundred ninety-nine dollars.

Norm: Oh. If it were a dollar more it would be way beyond my budget!

Holly: No problem! Here, I'll make a copy for you. I just need to find a blank disk. Here's one!

Norm: No, thanks, Holly—please don't.

Holly: Why not?

Norm: Because that's copyrighted software.

Holly: So?

Norm: You know what copyright is. Books, films, works of art—they're usually protected by copyright. It's illegal for anybody to make copies of the work without the copyright owner's permission. Software like SuperSoft Millenium is protected in the same way.

Holly: Aw, c'm'on, Norm—who's going to know it if we just make one little copy?

Norm: I'm not worried about going to jail. But it wouldn't be right. And you know we can't be happy unless we do the right thing.

Holly: Why wouldn't it be right? And what's that got to do with being happy?

Norm: It wouldn't be right because it would be acting contrary to human nature. And we can't be happy unless we act according to human nature.

Holly: I don't understand what you mean by "acting according to human nature."

Norm: Well, would you agree that we're social animals—that sociality is part of human nature?

Holly: Sure, that's right. There may have been a few "wild children" who grew up without any human contact, but they're definitely not normal. Even Mowgli, in the story, grew up with a wolf family. He wasn't all alone.

Norm: But isn't it also part of our nature that we're individuals, unlike bees and ants?

Holly: What do you mean?

Norm: Social insects act for the good of the hive. A bee, for example, will sting an enemy to protect the hive, even though it rips its own guts out and dies from the stinging. It doesn't have any conception of individual self-interest. But human beings act to achieve their own personal well-being, not just the well-being of the society they belong to. To be happy, we have to find a proper balance between the common good and the individual good. Nobody who's entirely self-centered can be happy—but neither can anyone who's not concerned for his own welfare.

For example, we want to work at an occupation that does good for our community, but we'd be unwilling to work without pay. Likewise, we'd like to have good things for ourselves, but we wouldn't want to steal them. All of this is entirely reasonable, and it's the way that we have to be happy because of this human nature that we have, which is both social and individual.

Holly: Okay, I agree with that.

Norm: If you agree with that, would you also agree that it's a consequence of this human nature that we must justly reward those who provide us with goods?

Holly: Oh, I get it! Because of the individual side of the nature of those who provide the goods—they don't want to spend their whole lives giving good things to others and getting nothing in return. And because of the social side of the nature of those who receive the goods—they don't want to be parasites on society, receiving, but giving nothing in return.

But what's this got to do with copying software? I mean, it's not as though we'd be *stealing* anything.

Norm: Yes, we would be stealing.

Holly: How can that be? If I steal a bicycle, I'm depriving the owner of that bike, but if I copy a piece of software, doesn't the owner still have it? I mean, the copyright owner has the original of the copied work, right, and I haven't exactly taken it away from him. So how is it stealing?

Norm: Well, what if you sneak into a theater or a ball game without buying a ticket? That would be stealing, wouldn't it?

Holly: Of course!

Norm: But what would you have stolen, actually? What would you have deprived the owner of? A view of the game? The sounds and sights of the show?

Holly: No. The owner still has those, if she ever had them. What the owner's deprived of, in that case, would be the income from the ticket.

Norm: Exactly! It's the compensation or reward that she should have for creating the show. Remember, you wouldn't want to work without pay. Neither do the folks who write software.

Holly: I guess I wouldn't want to deprive a mother of the bread she needs to support her children. But what about that Bill Doors, the owner of SuperSoft? Isn't he one of the richest men in the country? What's a few hundred dollars more or less to him?

Norm: No doubt Bill Doors has too much money. But I'm sure that if SuperSoft earned even more money, they would hire more mothers and fathers (and people without children too) to write more programs.

Holly: But, Norm, do people only write software for money? Don't some programmers write programs just because they love to do it, and then give them away for free?

Norm: That's true, Holly, and we have to distinguish between three kinds of program, at the money end.

Holly: What are they?

Norm: First, there's commercial software, like SuperSoft Millenium. You pay for the package up front, before you open it. The second kind is *shareware*. Shareware is usually made available on bulletin boards or on the Internet, where people can freely copy it. People can also copy it from the disks of other people who already have a copy. After they try it out, if they like the program enough to keep it, they're expected to pay a fee to the authors. Unlike commercial software companies, the authors of shareware don't try to prosecute those who don't pay; instead, they rely on the user's conscience. The third kind, *freeware*, is also called *public domain* software. Freeware is free software. It's distributed in the same ways as shareware, only without money. Sometimes freeware comes with a copyright notice—not so the author can require you to pay money, but so that he can license it in a way that makes it *illegal* for others to redistribute it for money.

Holly: Wait a minute. If there's all of this freeware out there, do we really need to pay software developers to encourage them to make software?

Norm: What do you think? Of the software you use every day, the software that's most important to you, which programs are commercially developed, and which are freeware?

Holly: Hmm, there's OakTree Ledger and of course SuperSoft Millenium. Also MudFlats PhotoFinish and Mediterranean Presentations. That's all commercial. Then on the freeware side, there's the Star Trek game—that's fun but not important. Oh, yeah, my World Wide Web browser is freeware, too. That's important, as well as fun.

Norm: Don't forget your operating system.

Holly: Won-Doze XCV! That's a lot of weight on the commercial side.

Norm: All of that commercial software wouldn't exist without the profit incentive. Unfortunately, copyright protection is not very effective. According to the Business Software Alliance, of the software sold in the United States, about 40% of the packages are illegal copies; in Japan and Germany, 75% are illegal, and in China, Korea, Taiwan, Spain, and Switzerland, 90% are illegal. The countries where the worst copying occurs are definitely *not* known for their flourishing domestic software industries.

Holly: No wonder!

1.1 Summary

We should not make use of the labors of others without rewarding them, unless they give us permission to do so. Our human nature as individual-but-social requires that those who produce social goods be adequately rewarded. Computer software and documents, as well as literary and artistic works in more traditional media such as paper, are social goods (if they are good at all), because they can easily be copied and widely distributed. These works are protected by copyrights because our government recognizes the need to reward their creators and so encourage them to continue creating such good works. By guaranteeing to those who created a work the exclusive right to produce it—with the implied opportunity to sell it to make a profit—the law encourages workers and their employers to produce works that people will value enough to pay money for. If we break the copyright law, we are withdrawing the incentive to produce works of this kind—including the work which, by the very act of copying, we show that we value.

2 A few other facts about copyright

1. Copyright obviously applies to electronic documents as well as programs—documents such as those found on the World Wide Web, on news groups, and email messages. And not only textual documents but also pictures, sounds, "movies," etc.

2. No special action is required to copyright a work. In particular, the author doesn't have to put a copyright notice on it or send a copy to the Library of Congress; it's automatically copyrighted. The copyright notice just makes it more obvious that the work is protected.

3. The prohibition of copying is not absolute. Obviously, the copyright owner can give you explicit permission to copy the work. Permission can also be given implicitly (e.g., when you write a letter to the editor of a newspaper, it's understood that the newspaper may publish it). Under certain conditions, copying is also permitted under the "fair use" doctrine of the copyright law. The intention of copyright is to give the owner what is reasonable, not absolute control.

3 Copyright of User Interfaces

Copyright applies to the form of expression, not to the ideas expressed. When applied to programs, this means that the source and object code (i.e., program instructions written by the programmer) are protected by copyright, but not the algorithms which the code expresses. What about other expressive aspects of the program, such as the menus, commands, keystrokes, etc., by which the user controls the program, and the displays, windows, etc., by which the program presents information to the user? These aspects of a program are called the *user interface*—the place where the user and the program interact. The "look and feel" of a program derives from its user interface.

Q. 1 What is the user interface of an automobile?

(Answer on page 10)

(Answer on page 11-??.)

Can a user interface, or the look and feel of a program, be protected by copyright? Copyright law has been in a very confusing, unsettled state regarding this issue.

Prior to about 1990, software developers felt free to copy the user interface of other programs (but not, of course, the programs' code). In 1990, Lotus Development Corporation began winning lawsuits against Paperback Software and Borland International, Inc., for copying the user interface of its spreadsheet program, Lotus 1-2-3. Apple Computer, Xerox Corporation, and other companies have filed similar suits.

Some companies even claim to have copyrighted programming languages. In the Borland case, the U. S. District Court of Massachusetts ruled that the hierarchy of menus used in Lotus 1-2-3's user interface was *protectable expression* under the copyright law, because Borland could have renamed the menu commands—for example, by changing "Quit" to "Exit" and "Copy" to "Duplicate." In March, 1995, the U. S. Appeals Court (First Circuit) reversed this decision, holding that the menu command hierarchy is a *method of operating* the program; methods of operation are specifically excluded from copyright protection, under Section 102 (b) of the United States copyright law. (The Appeals Court did not rule on whether other aspects of the user interface, such as the screen displays and the help prompts, were copyrightable.) Lotus appealed the case to the Supreme

Court, which recently (Jan/Feb '96) upheld the decision of the Appeals Court; i.e., Lotus lost, the Lotus menu command hierarchy is not copyrightable.

In our opinion, the decision made by the Appeals Court and the Supreme Court was wise: any monopoly (such as copyright) over a method of operation could have horrible consequences for users. Imagine, for example, that the first car manufacturer had a copyright over the automobile user interface that we're accustomed to today. Then other car manufacturers would either have to license the interface (if the first manufacturer would let them!) or invent a different interface. This would probably result in the development of dozens of incompatible interfaces for operating automobiles—some with steering wheels, some with sticks to control direction; some with brake pedals, some with hand levers for brakes. Imagine the confusion you would have if you tried to drive a different kind of car! Imagine the confusion there would be if every typewriter manufacturer used a different arrangement of the keyboard—which is nothing more than a menu of letters!

The situation is similar for computer users. If a new and better method of operating a computer program can be thought up, then that method should be available for all programs. Of course, the developer of the method should be adequately rewarded, but usually these rewards come from the sale of the first program to have this method of operation. Copyright law must balance the objectives of encouraging innovation, preserving competition, and rewarding the labor of authors; but the good of the public must come first, and this seems to require the free copying of methods of operation. On the other hand, some aspects of user interfaces, such as the phrasing of a help message and the layout of information displays, may be expressive, not operational, and as such would deserve copyright protection. (Note that the Appeals Court made no decision about any non-operational aspects of user interfaces.)

4 Patenting Computer Programs

Copyright is considered to be a form of "intellectual property right," a right of ownership over something of an intellectual nature—in this case, a certain form of expression.

The law recognizes other forms of intellectual property right, including trademarks, which apply to names of products or services, and patents, which apply to inventions.

Under U. S. law, in order to be patentable, an invention must fall under one of the following four categories: "process, machine, manufacture, or composition of matter." Inventions falling in these categories are said to have "statutory" or "patentable subject matter."

In addition, the invention must be novel, useful, and non-obvious. Patents require an application to the U. S. Patent and Trademark Office (USPTO).

The application process is expensive, typically costing several thousands of dollars in attorney's fees. If the patent is granted, it gives the inventor an exclusive right to the invention, lasting for seventeen years, even if somebody else independently reinvents the same thing.

What about patenting computer programs or algorithms? Traditionally, mathematical algorithms and laws of nature could not be patented. To patent an algorithm or law of nature would be to restrict the freedom of thought, and would restrain inventors excessively (since almost all inventions depend on laws of nature).

However, an *application* of a natural law could be patentable. For example, you could not patent the theory of general relativity, but you could patent a space ship that uses relativity to go faster than light, if such a thing were possible.

However, in Diamond v. Diehr (1981), the Supreme Court allowed a patent in which an algorithm was used by a computer to control a rubber-molding press, ruling that the presence of a computer algorithm in an otherwise patentable invention does not make that invention unpatentable.

Since Diamond v. Diehr, the USPTO has allowed many patents for computer programs and algorithms ("software patents").

Many computer programmers deplore this tendency. First, they argue that patent examiners were insufficiently trained in computer science and consequently gave many software patents for techniques that were too obvious. The fact that many software techniques are independently reinvented supports the claim that most of them are too obvious to deserve patent protection.

However, even if this is true, the USPTO could remedy the problem by hiring better qualified examiners for software patents. Mistakes in judgment do not argue a mistaken policy. Second, programmers argue that software patents are not needed to stimulate invention in the software industry, and they may even hinder it because many small software development shops would not be able to afford the expense of searching for a prior patent on a technique they want to use, and may be deterred by the threat of lawsuits for patent infringement. Many programs contain hundreds of techniques, each of which is a possible patent infringement if software patents are allowed, so the danger is real.

The Supreme Court has not ruled on whether computer algorithms are patentable subject matter, and is unwilling to do so, since a proper decision requires the broad investigative powers of Congress to hold hearings on the matter. However, the Court has held that "an algorithm, or mathematical formula, is like a law of nature, which cannot be the subject of a patent." (Diamond v. Diehr)

Yet the USPTO has taken the position that the non-patentability of algorithms "applies only to *mathematical* algorithms since any process is an 'algorithm' in the sense that it is a step-by-step procedure to arrive at a given result." (Barrett, 1989, emphasis added) We find this position almost incomprehensible, for two reasons.

First, "processes" (the kind that were traditionally patentable) transformed physical matter. Computer processes, however, are directed by programs toward the transformation of information, not matter. (It is true that matter is incidentally transformed, since the information is encoded in a material form; however, it is the information, not the matter, that counts.) Processes of *thinking*—mental steps—have long been recognized as unpatentable subject matter, but it is hard to find a distinction between "transoforming information" and "thinking."

Second, the USPTO appears to be making a distinction between mathematical algorithms, which are non-patentable, and computer algorithms, which (it holds) are patentable. The terms "mathematical algorithm" and "computer algorithm" are not strictly synonymous. For example, a person can use the algorithm for long division without the aid of a computer. So a mathematical algorithm is not necessarily a computer algorithm.

But isn't a computer algorithm necessarily also a mathematical algorithm?

Computer science grew largely out of mathematics, and it is hard to see any difference between computer algorithms and mathematical algorithms.

The only difference we can imagine is between mathematical algorithms dealing with numbers and algorithms for non-numerical problems. But to suppose that non-numerical problems are non-mathematical is to take an extremely narrow view of mathematics. For example, boolean logic and geometry are branches of mathematics, yet they do not deal with numbers.

Computer algorithms also deal with text (e.g., word processing), which seems to be a non-numerical problem, but text is represented in the computer in the form of numerical codes.

In fact, *all* information in the computer is represented by binary numbers. The USPTO appears to be making an artificial distinction between mathematical algorithms and supposedly non-mathematical computer algorithms. That distinction has no basis in reality.

If one cannot patent an algorithm, can one patent the *use of a computer* to carry out the algorithm? This would be either a patent on a process or on a machine, so it seems to qualify as patentable subject matter. However, a patentable invention must also be non-obvious.

Given any newly discovered algorithm, the decision to use a computer to "run" the algorithm is now a matter of routine. The use of a computer to carry out an algorithm may have been non-obvious in the 1940's, but it is very, very, very obvious by now.

The best argument against software patents appears to be the fact that the software industry has flourished for many years *without* the "protection" of patents.

Recently, a number of large companies have been trying to get as many software patents as possible. This proliferation of software patents may cause smaller software developers to go bust. To defend against a lawsuit for patent infringement, even where the patent is obviously invalid, may cost hundreds of thousands of dollars. The industry doesn't need patent protection; the most innovative (typically smaller) software companies will almost certainly be hurt by it. The purpose of the patent law is to stimulate invention, not to wither it up.

5 Exercises/Problems for Discussion

1. A college's director of computing services pays for ten copies of a program which she expects will be used by at most ten persons at any one time, but installs the program on one hundred computers. Comment on the moral and legal propriety.

2. Richard Stallman believes that the Golden Rule obliges him to give away the software he finds to be useful. The law won't allow him to do this with software copyrighted by others, so he and his associates write similar software themselves and then give it away. Read Stallman's "The GNU Manifesto" (http://www.gnu.ai.mit.edu/gnu/manifesto.html) and comment on Stallman's belief and practice.

   Also see Stallman's "What Is Copyleft?"[1] with link to the GNU General Public License, and "Why Software Should Not Have Owners"[2]

[1] http://www.gnu.ai.mit.edu/copyleft/copyleft.html
[2] http://sugwww.uni-paderborn.de/Docs/fsf1.html

6 Philosphical Note

The way we think of human nature has a powerful impact on what we think is right and wrong and what we think is a happy life. For example, if we believe that human nature is basically selfless, as Marx did, then we'll probably favor some form of socialism, in which there is no profit incentive to produce high-quality software. But we know that socialism doesn't work, because human nature is not entirely selfless. Marx grasped only one side of human nature, the social side; he failed to see the individualistic side. The English philosopher Thomas Hobbes (1588–1679) fell into the opposite error. He saw human nature as basically individualistic and selfish. According to Hobbes, civil society is a convenient arrangement which we've invented to prevent ourselves from killing each other off, but it's not natural for human beings to live in society or to restrain their egoistic impulses. It's natural for each individual to attack and plunder his neighbors, to take software and other goods from them by cunning or by force. It's hard to see how anyone, on such a conception of human nature, could be happy living under the restraints of civil society. Fortunately, that conception is wrong.

Norm's ethical theory, which makes a strong connection between moral values, human happiness, and human nature, is called "ethical naturalism" or sometimes "natural law" theory. Its roots are in the philosophies of Plato and Aristotle.

6.1 Optional Philosophical Exercises/Problems for Discussion

1. Ethical naturalism isn't the only ethical theory that philosophers have come up with. Two others are Kantianism and utilitarianism. According to Immanuel Kant (1724–1804), we should act only according to those maxims (principles) which we can will to be universally practiced. For example, I should not tell a lie to avoid some embarrassment, because the maxim "It is okay to tell a lie to avoid embarrassment" is one that I cannot wish everyone would follow. Comment on violating copyright law from the Kantian perspective.

2. According to the utilitarian theory, morally right actions are those that maximize the total utility, or balance of pleasure over pain, for the entire community. For example, murder would be wrong because the pain inflicted on the victim and those who loved him outweighs whatever pleasure the killer gets. Comment on violating copyright law from the utilitarian perspective.

7 References

The League for Programming Freedom[3] has two position papers, "Against User Interface Copyright" and "Against Software Patents," and other information.

Norm's figures about illegally copied software are taken from Tom Forester and Perry Morrison, *Computer Ethics*, MIT Press, second edition, 1994, p. 52.

The classical source for Kant's ethical theory is his *Foundations of the Metaphysics of Morals*, also translated with a variety of other titles. It is a very small book, but not easy reading.

[3] `http://www.lpf.org/`

Kant's *Lectures on Ethics* (Harper Torchbacks, 1963) is a more readable introduction to his ethical thought; it is a reconstruction of his lectures based on notes taken by three students.

John Stuart Mill's *Utilitarianism* is an excellent introduction to utilitarian ethics.

Aristotle's major ethical work was the *Nicomachean Ethics.* Plato wrote many dialogs on ethical problems; one of the most famous and systematic, from the point of view of relating moral values to human nature, is the *Republic.*

Other references:

Barrett, Lee E. (1989). Untitled paper[4] on the patentability of mathematical algorithms and computer programs.

Davis, Randall, Pamela Samuelson, Mitchell Kapor, and Jerome Reichman (1996). "A New View of Intellectual Property and Software," *Communications of the ACM,* vol. 39, no. 3, March 1996, pp. 21-30.

Diamond v. Diehr, 450 U.S. 175 (1981).

Hobbes, Thomas (1651). Leviathan.

Johnson, Deborah G. (1985). *Computer Ethics*, Prentice-Hall.

Lotus v. Borland, (1995).[5]

8 Answer to Question

The user interface includes the steering wheel, accelerator, brake pedal, gear selector, speedometer, fuel gauge, and other devices which the driver uses to control the automobile.

[4] `http://www.lpf.org/Patents/papers/barrett.paper`

[5] `http://www.lpf.org/Copyright/lotus-v-borland.html`