COPYRIGHT PROTECTION FOR SOFTWARE

Ralph Oman[*]

The United States blazed the trail in giving copyright protection for software. Until just recently, many other countries favored *sui generis* protection for software, but that argument was finally settled in GATT/TRIPs and last December's World Intellectual Property Organization Copyright Treaty. Copyright is now universally seen as the preferred means of protection. To try to make certain that we all have a clear idea of the metes and bounds of protection for computer software in the United States, let me start with a few copyright basics.

Copyright protects the authors of original works of authorship. You know what they are: sculpture, novels, poems, paintings, newspapers, newsletters, jewelry, fabric designs, recipe books, motion pictures, sound recordings, maps and charts, architectural works, cartoons — the list goes on and on. The copyright law does not normally protect useful articles. Generally the patent law protects useful articles. Copyright protects a lamp base in the shape of a Balinese dancer that is artistic. But it doesn't protect the design of the lamp as a whole. Even so, copyright in fact has always protected useful works. Maps and charts have been protected since 1790, so it comes as no surprise that copyright also protects computer programs, very useful creations that are essentially operating instructions for a machine. The courts have played a major role in defining the scope of copyright protection. After deciding the basic issues of *copyright-ability* of software, they got into the tough issues. Under United States law, computer programs are literary works. As with other literary works, the law protects both literal and non-literal features of a program.[1]

---

* Counsel, Dechert Price & Rhoads; Pravel Professorial Lecturer in Intellectual Property and Patent Law, George Washington Law School. Former United States Register of Copyrights; Chair, American Bar Association's Committee on International Copyright Treaties and Law.

1. *Literal* refers to the actual source code or object code or the computer screens or the user interfaces, and *non-literal* refers to the SSO, the plot, the flow of elements one into another, and the relationship of the elements one to another.

I. THORNY ISSUES

But it's not quite so simple. What's protected? What's copyrightable? What's not, particularly on the non-literal side?

Copyright, of course, protects only the *expression* of ideas, not the ideas themselves. In copyright cases, the defense often claims that it has only borrowed un-protectable ideas, rather than protectable expression. In *Morrissey v. Procter & Gamble*, a 1967 non-software case involving written instructions for entering a promotional contest, the circuit court stated the general principle: if a work is so simple and so straightforward as to leave available only a severely limited number of ways to say something, the expression would be un-copyrightable, even if it was very creative.[2] The idea and the expression had *merged*. Since we use computer programs in a functional context, the idea/expression argument is often transformed into an inquiry as to whether or not copyright in a program gives the copyright owner a monopoly over a technological function.

In an early series of cases, *Whalen v. Jaslow*, the most famous, the courts developed a reasonably simple approach to this issue.[3] To see if somebody had copied expression or ideas, the judges determined whether or not other programs could be written that performed the same function as the copyrighted program. If another program could be written to perform the same function, then that program is an *expression* of the idea and protected from copying. The idea is very general: in this case, the organization of a dental office. Everything else is expression, including SSO. Of course independent creation of an identical program is okay. This simple approach has not survived, particularly in the most difficult area of the law trying to figure out if somebody infringed not the actual computer code, the *literal* aspects of the program, but the non-literal aspects, the SSO.

In 1997, the Court of Appeals in New York decided *Computer Associates v. Altai*. The case deals with the question of whether the scope of protection of the *non-literal* aspects of a computer program may be protected by copyright.[4] What is an *idea* and what is *expression*?

The decision rejects the broad approach I just described. The *Altai* court declined to find infringement even when faced with strong evidence of copying of non-literal elements. The defendant, Altai, had admitted copying the actual code of one version of the plaintiff's program and paid

2. Morrissey v. Procter & Gamble, 262 F. Supp. 737 (D. Mass. 1967).

3. Whalen v. Jaslow, 609 F. Supp. 1307 (E.D. Pa. 1985).

4. Computer Assoc. v. Altai, 126 F.3d 365 (6th Cir. 1997).

damages of $350,000. The real dispute concerned a second, so-called *clean* version of the program that Altai programmers created without seeing all of plaintiff's source code. For this *clean* room version, the appeals court found that there was no copying of the literal computer code. The court then looked for copying of the structural or organizational (the non-literal) elements of the program.

It looked for substantial similarity between the non-literal elements of both programs. Substantial similarity, of course, is the standard the courts apply in finding whether or not infringement has taken place. It found none. Any SSO that was very similar was not copyright infringement. In its analysis, the court applied what we called the *abstraction test* to determine whether or not the non-literal aspects of computer programs are substantially similar. The court also drew on the doctrines of merger, *scnes'-faire*, and public domain, which I will explain.

Under the *merger doctrine*, of course, since the expression is inseparable from the idea, you can not get protection. The *scnes'-faire* doctrine holds that certain stock or standard literary devices are not copyrightable.

Applying these doctrines, and the principles of non-protection for public domain elements, the court reached its most critical conclusion regarding the similarity between the plaintiff's and defendant's programs. The court said that the similarity in the structure between the plaintiff's and Defendant's program was dictated by the nature of other programs with which it was designed to interact, and thus, is not protected by copyright. The court did not ask the tough question: Is it independently created or does the similarity result from copying?

It is important to note, however, that the *Altai* court accepts the principle that copyright protection can extend to a computer program's non-literal structure. The amount of protection due structural elements, in any given case, will vary according to the protectible expression found in the program at issue.

Let's see how the *Altai* court applies the abstraction test. Step-by-step, it analyzes both programs in order of increasing generality from object code, to source code, to parameter lists, to services required, to general outline. Object code and source codes are protected as literal elements. Then the court moves further down the spectrum of abstraction and as it goes it filters out protectible expression from non-protectible elements to determine the scope of the plaintiff's copyright in the non-literal structure of a computer program. The court filters out the elements dictated by efficiency, function, the programming techniques that all programmers use, external factors (interoperability), or elements taken from the public domain. Finally, the court compares the remaining

protectible expression in Computer Associates' program with Altai's program to see whether or not the defendant copied any aspect of protected expression, any of the remaining golden nuggets.

Using this three-step abstraction-filtration-comparison test, the *Altai* court has a much narrower view of exactly what components of the program are subject to copyright protection. Under this test, quite a bit of copying is tolerated, perhaps more than would be allowed in true literary work. With a clearer idea of what is protected, generally speaking, the source code, object code, the golden nuggets of the program's non-literal aspects, let's look at a few related controversies. These too, are literal aspects of a computer program. We see them and hear them and touch them on the screen.

A battle arose over the protectibility of screen displays and other *user interfaces*. As with other works, to be protected, computer screens must contain more than *de minimis* copyrightable authorship. Some computer screens only record information, and they are often not copyrightable because they are just blank forms, or just lists of common words, and lack enough original expression to support a claim to copyright. Even so, in 1993 a district court in Boston, in *Lotus v. Borland*, found that a menu tree contained enough originality to be copyrightable.[5] Even though functional considerations played a part in the creation of the menu, the court found that function did not dictate the final version of Lotus' menu on the screen. The court pointed out that a great variety of possible words and phrases could accomplish the desired function. The court gave three reasons for its finding. First, Lotus' format depends on the programmer's personal judgments and preferences among many possible choices. Second, even the user of the program can change the menu tree, so how can it be dictated by function? And third, the court noted that many other spreadsheet programs used different menu trees, and mere functionality did not account for these differences. In conclusion, the court found that Borland's menu tree was sufficiently similar to Lotus' to constitute copyright infringement.

That decision did not survive the appeal. In March of 1995 the First Circuit overturned the district court's decision and held that Lotus' menu tree, made up of words and phrases, is un-copyrightable subject matter as a matter of law. Citing Section 102(b) the 1976 Copyright Act, the court found that textual menus (as opposed to complex graphic or animated user interfaces) are simply a *method of operation*, for which Section 102 explicitly prohibits copyright protection.[6] The court explained,

5. Lotus v. Borland, 831 F. Supp. 223 (D. Mass 1993).

6. Pub. L. No. 94-553, 90 Stat. 2541-2602 (1976).

"we think that method of operation . . . refers to the means by which a person operates something, whether it be a car, a food processor or a computer . . . ."[7] In many ways, the Lotus menu command hierarchy is like the buttons used to control, say, a videocassette recorder, or like the dashboard of a car.

In a 4-4 decision last year, the Supreme Court upheld the First Circuit's decision, without an opinion that would have helped clarify the law.

Enough on copyrightability. Let me discuss one last controversy:

II. REVERSE ENGINEERING

A very significant issue on the infringement side is whether or not someone can reverse engineer a copyrighted program to produce a competing program. Let me explain. By reverse engineering, somebody can figure out the physical composition or electrical properties of electronic, mechanical, chemical, and other industrial products. As applied to computer programs, reverse engineering refers to the whole range of activities, from the study of publicly available sources of information about a program to the process of creating *pseudo-source code*, as well as *decompilation* or *disassembly*, breaking down the program to its component parts and then rebuilding it sentence by sentence.

We have to keep coming back to the same basic premise, copyright protects expression, and not ideas. Copyright does not protect the functionality of a program. Nothing in the copyright law prevents someone from analyzing program code, then taking the ideas, algorithms, or methods used in the program to create another program.

Anyway, the reconstruction of the original source code from the object code is like doing a puzzle. You use a decompiler or disassembly program to search the original for known or anticipated instructions. One method used to separate idea from expression is the so-called *clean room* approach used by Altai. With this approach, you would attempt to extract only the ideas from a competing program in order to replicate its functions. A *dirty room* team actually *copies* the original program and decompiles it to develop a pseudo-source code. The team studies the code to identify interfaces and document ideas. They then prepare detailed written descriptions of the design elements of the original program without using actual code, and programmers in the *clean room* take that intermediate product, that detailed script, and work from its description to imitate the original program. One problem in this approach is that if too much actual

7. Lotus v. Borland, 73 F.3d 355 (1st Cir. 1995).

detail from the original program gets into the *clean room*, even structure, sequence and organization, the non-literal elements, they may wind up scrapping the end product as *too dirty*, or *too infringing*.

But the basic question remains: does decompilation appropriate more than unprotected ideas in the attempt to accomplish the same functions of another program? Decompilation does involve the copying of a computer program if only as an intermediate step and, is, therefore, a *prima facie* infringement. The primary rebuttal argument relies on the fair use defense, codified in Section 107 of the United States copyright law. Decompilation for academic research, such as a computer science professor performing classroom analysis with his students is, in all probability, within the fair use privilege. Decompilation for commercial purposes normally stands on a different footing. Copyright owners argue that the decompilation of a program to produce a competing product fails all four of the fair use factors:

1) the nature and purpose of the use is entirely commercial;
2) the copyrighted source code, as an unpublished work, is subject to a very narrow scope of fair use;
3) the entire work is copied;
4) harm to the market for the original is presumed with commercial use.

Decompilers reject this claim. They say their purpose which is to gain access to ideas, is a socially valuable one. They argue that software is the first and only copyrightable work that is not transparent, that is not read or played when it is used, and as such, does not clearly reveal its *ideas* or *expression*. Since copyright does not protect ideas, the argument goes, they should be available to the public, and decompilation is one of the few ways to accomplish this. In *Lotus* v. *Borland*, the First Circuit discussed the economic implications of interoperability, and concluded that software compatibility has a beneficial public impact that should be encouraged.[8]

Decompilers also argue that the market factor weighs in their favor since the end result is non-infringing; any market loss is attributable to the appropriation of idea, not expression, and to the building of a better product. They may have an unfair competitive advantage since they did not have to pay for the original innovative development costs.

On the other hand, others point out that although most copyrighted works disclose their ideas on inspection, this is not a requirement, since copyright protects unpublished works. And, Copyright Office regulations

8. *Id.*

let people deposit copies of programs with the trade secret portions blocked out, which blocks access to ideas as well as expression.

So we have reached the point in our history where we have more questions than answers. How have the courts resolved the related issue of interoperability? Let's look at one case decided by the Court of Appeals for the Federal Circuit.

Judge Rader of the Federal Circuit in *Nintendo v. Atari*, found that the unlocking program contained protectible expression. He affirmed the lower court's holding that Nintendo would likely establish that Atari infringed its locking program by copying the literal elements of the source code. However, Judge Rader noted an important qualification. He specifically reversed the lower court's finding that Atari's intermediate copying of the locking program for the purpose of reverse engineering infringed Nintendo's copyright. The court found such intermediate copying was *fair use:* in Judge Rader's words, "[r]everse engineering object code to discern the unprotectible ideas in a computer program is a fair use."[9] Of course, the court did not say that the fair use doctrine authorizes unrestrained reverse engineering. One can reproduce the software only to the extent necessary to understand uncopyrightable portions of the work. In the words of Judge Rader, any reproduction of protectible expression must be strictly necessary to ascertain the bounds of protected information within the work.

So I have given you the basics and the hot issues. Winston Churchill once said that democracy is the worst form of government, except for all the others. In many ways, copyright is the worst form of protection for software, except for all the others. Although patent protection shows some limited promise for break through ideas, copyright will continue as the primary means of protecting software for the foreseeable future.

9. Atari v. Nintendo, 1993 WL214886 (N.D. Cal. 1993).