COPYRIGHT PROTECTION FOR SOFTWARE

Ralph Oman

I. THORNY ISSUES.......................................................... 340
II. REVERSE ENGINEERING.............................................. 343

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

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. 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. 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. 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

---

Oman

损坏的$350,000。实际的争端涉及第二个，所谓的“清洁”版本的程序，Altai程序员创建的，没有看到原告的源代码。对于这个“清洁”版本，上诉法院发现没有对字面计算机代码的抄袭。法院然后寻找对字面元素的抄袭。法院发现没有抄袭。任何非常相似的SSO都不构成版权侵权。在分析中，法院应用了我们所说的抽象测试来确定计算机程序的非字面元素是否实质性相似。法院还引用了合并、scenes'-faire和公共领域的教义，我将在下面解释。

根据合并教义，当然，因为表达是想法不可分的，所以不能受到保护。scenes'-faire教义认为某些固定或标准的文学装置不具有版权保护。应用这些教义和非保护原则，法院对原告与被告的程序之间的相似性做出了最重要的结论。法院表示，原告和被告的程序之间的结构相似性是由其设计要与之交互的其他程序的性质所决定的，因此，不受版权保护。法院没有问一个更艰难的问题：是独立创造的还是相似性结果于抄袭？

然而，值得注意的是，Altai法院接受了一个原则，即版权保护可以扩展到计算机程序的非字面结构。在任何给定的情况下，对结构元素的保护将根据所涉及程序中可保护的表达形式而变化。

让我们看看Altai法院如何应用抽象测试。一步一步地，法院分析了两个程序，从对象代码到源代码，到参数列表，到服务所需，到一般大纲。对象代码和源代码被看作是字面元素。然后法院进一步地将抽象性下降，对非保护性因素进行过滤，以确定原告对程序非字面结构的版权。法院过滤掉了那些由效率、功能、程序员都使用的编程技术、外部因素（可交互性）或来自公共领域的因素。法院比较剩下的元素。

它是一个重要的问题，但是，Altai法院接受了一个原则，即版权保护可以扩展到计算机程序的非字面结构。这种保护将根据程序的抽象性水平而变化。法院对非保护性因素进行过滤，以确定程序非字面结构的版权。法院比较剩下的元素。
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. 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. The court explained,

"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 . . . ." 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

---

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 unprotected ideas in a computer program is a fair use.” 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.

---