

The Role of Intellectual Property in Open Source Software^{*}

Dr. Lee A. Hollaar
Professor, School of Computing
University of Utah
hollaar@cs.utah.edu
www.cs.utah.edu/~hollaar

One might expect that “open source” software would be a good fit for a country without strong intellectual property laws. After all, the software developers are making the source code for their programs available to the public at no cost, and allowing its further redistribution.

However, there are substantial differences between “open source” or “free” software and software that is actually in the public domain, without any form of intellectual property protection. Both “open source” and “free” software depend on licenses, and the threat of legal action based on copyright infringement or breach of contract, to impose requirements on downstream distributions of the software and its modifications. That requires strong intellectual property and contract laws for those requirements to be more than just nice ideas.

But those same strong intellectual property laws can also hamper the reimplementing of a proprietary program as open source. Any country wishing to create an environment for open source development needs to be very careful in structuring its intellectual property laws.

Open Source and Downstream Requirements

The ideals of both “open source” and “free” software permit anybody to modify and redistribute that software, but that is substantially different than placing the software in the “public domain.” Rather than impose no restrictions, as would be the case if the software were really in the public domain, open source software licenses generally require that any redistribution of the modified software be under the same terms as the original license.

The Open Source Definition¹ covers this in Section 3, “Derived Works”:

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

Rationale: The mere ability to read source isn't enough to support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications.

^{*} Copyright © 2004 by Lee A. Hollaar. Permission is granted to reproduce this paper in its entirety.

¹ <http://www.opensource.org/docs/definition.php>

For “free” software (as that term is used by the Free Software Foundation), this is most commonly expressed in the General Public License (GPL).²

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions: . . .

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License. . . .

(In the following discussion, the term “open source” will also include “free” software, unless it is clearly indicated. The term “proprietary” will be used to indicate software whose source code is not distributed or where there is no permission for modifications given, such as with most software marketed by software companies like Microsoft.)

This represents the *quid pro quo* for open source software – that you may use it as the basis for your implementation or modifications, but those modifications must be available to the community on the same terms as the original software. A company cannot take the work of an open source developer, include it in a program, and distribute it as a proprietary program whose source code is not provided or without the downstream right to modify and redistribute.³

The Need for Strong IP Laws

For the open source rules to have any meaning, there must be both strong copyright and contract law. Without a strong copyright law that gives the owner of the copyright of a work the right to control the distribution of derivative works, there is not need for an open source license to permit such a distribution. One does not need the permission of a license to do what one is already permitted to do. Without strong contract law, the special terms imposed by an open source license on any distribution are not easily enforced.

As an example of the problems that weak intellectual property laws can cause for open source licenses, consider the effort NASA has made to develop the “NASA Open Source Agreement.”⁴ Under United States copyright law, software produced by NASA employees cannot be protected by copyright.⁵ Instead, it attempts to use contract law to impose obligations on the recipient of the software similar to those of a conventional open source license.

But while somebody who violates the terms of a conventional open source license can be sued for copyright infringement, that is not the case with the NASA license, since NASA owns no copyright on the software. At best, the violator can be sued for breach of contract.

² <http://www.fsf.org/copyleft/gpl.html>

³ In the religious debates over “open source,” and particularly “free” software, sometimes this inclusion of another developer’s software into a proprietary program gets referred to as “making the open source software proprietary.” But that is not what happens. The original open source program is still available on the same terms as it originally was. The only thing that may be unavailable is the specific proprietary modifications that were made.

⁴ http://opensource.arc.nasa.gov/pdf/NASA_Open_Source_Agreement_1.3.txt

⁵ “Copyright protection under this title is not available for any work of the United States Government, but the United States Government is not precluded from receiving and holding copyrights transferred to it by assignment, bequest, or otherwise.” 17 U.S.C. §105.

And if the software was obtained outside of the license, such as through a Freedom of Information Act request or perhaps from someone in violation of the license, there is no license between NASA and the party obtaining the software, so the party is free ignore the terms of the NASA license.

The GPL explicitly recognizes this need for both strong copyright and contract laws to achieve its goals:

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Similar to the Needs for Proprietary Software

But strong copyright protection and the recognition in contract law of licenses that accompany software are also what protects proprietary software. Both want to have the terms of their license enforceable under the law, and to use copyright law to cover people who may not be party to the license or as the reason that a license is necessary. Both use the license to disclaim as many warranties as possible.

Of course, there are differences in the license terms between open source and proprietary software. Proprietary software licenses are generally concerned with the way that the software may be used (on a single machine, limited number of backups, no reverse engineering, etc.) while open source licenses are generally concerned with the modification and distribution of the software. But both attempt to control user behavior by license terms in trade for permitting what would otherwise be a copyright infringement. (Distribution in the case of open source; reproduction necessary to use the software in the case of proprietary programs.⁶)

It may seem strange to talk about how the legal structures needed for open source also provide the foundation for trade secret protection, because open source is the antithesis of trade secrets. But the foundation for trade secret protection – other than never disclosing the software to anybody – is contracts that govern what a person receiving a trade secret may do with that information. And the strong contract law that is necessary for open source is the same law that supports trade secret agreements.

But Not Too Strong

But one has to be careful not to have copyright laws that are too strong. Many open source programs are based on existing proprietary programs. The Free Software Foundation's GNU project is trying to create a "free" version of the Unix operating system, which was proprietary to Bell Labs at the time the GNU project started, and was only

⁶ In this respect, the proprietary software companies are also trying to use their "license" as an end run around 17 U.S.C. 117, which permits the "owner of a copy of a computer program" to make the incidental copies necessary to run a computer program. The license claims to make the purchaser of the software not its owner, but merely a licensee. The CONTU Final Report had originally suggested granting the right to any "rightful possessor of a copy" but Congress, with no explanation, substituted the current language. In *MAI v. Peak* (991 F.2d 511, 26 USPQ2d 1458, CA9 1993), the Ninth Circuit noted that licensees "do not qualify as 'owners' of the software and are not eligible for protection under Section 117." One possible reason is that Congress was concerned about extending this right to renters of computer software, who could legally make copies and then conveniently forget to delete them, but changes to 17 U.S.C. 109 in 1990 eliminated that reason.

available under a signed license with terms too restrictive for the Free Software Foundation's founder. (The name GNU is a recursive acronym "GNU is Not Unix," although it was trying to be.) Linux (or GNU/Linux, as the Free Software Foundation would like it) is the kernel that the GNU project seemed unable to get in place, and when combined with the other GNU programs gives an excellent reimplement of Unix.

There are two areas where open source development of programs to replace existing proprietary programs can run into difficulties: the copying of the nonliteral aspects of the existing program, and when a new program is a derivative work of an existing program. Creating new software from old is discussed in more detail in the Appendix to this paper, which is reprinted from my treatise *Legal Protection of Digital Information*.

In the United States, the abstraction-filtration-comparison test is used to determine which nonliteral aspects of a computer program are protected and whether there has been an infringement. The test is not well-specified and requires a considerable effort to apply, but there are a number of reasons why an element of a program is not protectable by copyright:

- The element's expression was dictated by reasons of efficiency, such as when it is the best way of performing a particular function.
- The element's expression was dictated by external factors, such as using an existing file format or interoperating with another program.
- The element's expression is a conventional way of writing something in the particular programming language or machine running the program.
- The element, at the particular level of abstraction, is an unprotectable process and not protectable expression.
- The element is taken from the public domain or is an unprotectable fact.

Copyright laws that do not provide an exception for expression dictated by external factors, for example, can make it impossible to create a new work which is based on a preexisting work, particularly if it is necessary to be compatible with file formats, as would be the case for an open source word processor that needs to be able to read existing Microsoft Word documents for people to adopt it.

What About Patents?

Probably the most contentious area of intellectual property when it comes to open source is patents. The Free Software Foundation is adamantly against patents on software-based inventions, as indicated in the GPL Preamble:

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

This position should not be surprising, since the original goal of the Free Software Foundation's GNU project was to create a "free" version of the Unix operating system. While it is possible to use techniques such as "clean rooms" to assure that the copyright of an existing program is not violated when it is reimplemented by somebody, there is no similar way to protect infringing a patent that may cover some critical aspect of a program. Independent development is not a defense to patent infringement.

This is more than a hypothetical argument. United States Patent 4,135,240, issued in 1973, covers a key aspect of the Unix operating system – the ability to set the identity of the effective user for a program based on control information in the program’s directory entry. This involves an additional bit (called SUID) that indicates that the effective user should be the owner of the program file, not the current user. This allows users to run selected programs as if they were a system administrator, and is used for many important Unix functions. It would be impossible to implement an operating system fully-compatible with Unix without infringing this patent.

Luckily for the GNU project, the patent owner (Bell Labs) dedicated the patent to the public shortly after it had issued,⁷ but the lesson was clear. Just one patent on a key aspect of a program could prevent the development of a “free” version of that program.

The Open Source Definition takes no position on patents.

To the extent that open source software remains primarily the reimplementations of existing proprietary software, patents will remain a real problem, and one even more critical since software companies like Microsoft have discovered software patents and have been building their portfolios.⁸ These patents can be easy to get if one doesn’t want breadth in coverage, but instead will be content with a patent that covers only a specific feature so that a program cannot be exactly reimplemented. One Microsoft patent (6,286,013) is so narrow that some of its claims recite the specific function codes used by Microsoft DOS or Windows. But it would be impossible to write an operating system capable of running Windows programs without infringing this patent.

But as open source developers move from reimplementing proprietary programs to developing new programs and techniques, patents may be the only way to prevent proprietary software developers from free-riding by reading the open source code to learn the techniques, and then doing their own implementation that does not infringe the copyright on the open source program.

Many companies also have extensive patent portfolios so that when they seem to be blocked by a patent, they have something that might be cross-licensed to get permission to use the blocking patent. If open source developers want to fight patents that may block their activities, they may want to develop patent portfolios of their own, and perhaps pool their patents toward the common good.

Summary

Any country that is interested in supporting open source software must develop strong intellectual property protection. The licenses vital to open source software depend on copyright and contract law. And while the “free” software movement is rabidly against patents on software-based inventions, such patents may be necessary to counter the patent portfolios of proprietary software companies.

⁷ It presumably was one of the many patent applications that Bell Labs had filed to test the boundaries of software patentability.

⁸ As of the beginning of May 2004, Microsoft held over three thousand patents.

Appendix

This appendix is reprinted from *Legal Protection of Digital Information*, published by BNA Books. Copyright © 2002 by Lee A. Hollaar. The full text of the treatise, along with the material it references, is available on the Internet at no cost at <http://digital-law-online.info>.

VI.D. New Software from Old

Most computer programs are based to some degree on other computer programs. They could be new implementations of existing programs (for instance, Linux reimplementing Unix) or new programs influenced by an existing program (the first spreadsheet program, VisiCalc, influenced Lotus 1-2-3, which influenced Microsoft Excel), or they could use a preexisting library as part of the final computer program.

Obviously, there are a number of copyright considerations when creating a new computer program based on an existing one.

VI.D.1. Using a Clean Room

It is not necessary to be looking at an existing program while writing a new program for the new program to infringe the reproduction right in the existing program. The copying could be unconscious. In *ABKCO Music v. Harrisongs Music*,¹⁰⁵ George Harrison was found to have infringed the copyright of “He’s So Fine,” a song that he had heard years before, when he wrote “My Sweet Lord.” If you have had access to the source of a computer program, you need to be particularly concerned that you aren’t unconsciously copying the original program when you write a similar program.

One way to avoid infringement when writing a program that is similar to another program is through the use of a “clean room” procedure. This is what was done when companies cloned the BIOS of the IBM personal computer to produce compatible systems. In a clean room procedure, there are two separate teams working on the development of the new program.

The first team determines how the original program works, by examining its source code if it is available (IBM published the source code for its BIOS in a technical manual), by reverse engineering the program (by converting its object code back to source code and attempting to understand it or by testing it to see how it behaves), or by studying available user manuals and other descriptions of the program’s function. This first team puts together a complete technical specification that describes the functioning of the original program. Such a specification is not an infringement, since the copyright in the original program doesn’t protect its functionality, only the expression in the program that creates that functionality. Generally, an intellectual property attorney will review the functional specification to assure that it does not contain any protected expression from the original program.

Given the functional specification, a second team of programmers, metaphorically in a “clean room” uncontaminated by the original program, implements the new program. These programmers have not seen the source code of the original program. In fact, it is best if they have never seen any aspect of the original program, getting all their knowledge of it from

¹⁰⁵ 722 F.2d 988, 221 USPQ 490 (2d Cir. 1983).

the functional specification. Because they haven't seen the original program, they cannot be copying it, even unconsciously.

A limited clean room was used by programmers at Altai when they discovered that one of their employees had written a program that included portions of a program he had worked on at a competitor. Although *Computer Associates v. Altai*¹⁰⁶ does not spend much time on the clean room aspects of Altai's new implementation, it does suggest that such a procedure results in a program that does not infringe as long as the portions that are similar are dictated by function.

VI.D.2. Piecewise Reimplementation

Many people have reimplemented computer programs by rewriting them to replace the source code with code of their own writing. There is no reason to believe that this would not be a copyright infringement, particularly if the reimplementer had access to the source code of the original program, even if none of the original source code remains.

When the first segment of code is rewritten, the new code will be an infringing work if it is substantially similar to the original code, or may be an infringing derivative work if it is a reimplementation in a different programming language. That reimplemented first segment is combined with the remaining parts of the original program to form an intermediate version. Subsequent modifications produce another work. So when you have completed the piecewise reimplementation, you have a set of works, each of whose creation infringes the exclusive rights of the owner of the copyright of the original program.

As an analogy, consider the translation of a novel to a different language, something that would clearly be a derivative work. It makes little difference that none of the original words remain, or that the translation was done a little at a time. The resulting translation is still an infringing derivative work.

Even if you completely replace the program with new code, nonliteral elements also protected by the original program's copyright are likely to remain and infringe – elements like the overall program structure or architecture and data structures that are not dictated by external or efficiency considerations. Although there is no case law on this point, it would seem that the only way to break the chain of infringing works is by some extraordinary act, such as a clean room implementation.

VI.D.3. Section 117 Adaptations

Most computer programs are covered by a series of copyrights, each coming into being when a portion of the program is written or modified. The copyright on each nontrivial modification is as a derivative work of some preexistent program. One of the exclusive rights of a copyright owner is the right to control the preparation of any derivative works, so generally only somebody with the permission of the copyright owner can modify a computer program.

There is a special exception in Section 117 that permits the owner of a copy of a computer program “to make or authorize the making of another copy or adaptation of that computer program provided that such a new copy or adaptation is created as an essential step in the utilization of the computer program in conjunction with a machine and that it is used in no other manner.”¹⁰⁷ This exception recognizes that it is sometimes necessary to

¹⁰⁶ 982 F.2d 693, 23 USPQ2d 1241 (2d Cir. 1992).

¹⁰⁷ 17 U.S.C. §117.

configure or otherwise modify a computer program as part of its installation or to run it. While in theory this adaptation exception would allow somebody to redo Microsoft Access to run on a Macintosh, in reality such an adaptation even with access to the source code would be impractical for an individual user.

Note that Section 117 also requires the permission of the computer program's copyright owner to transfer the adaptations you have made. So Section 117's adaptation right is personal to each owner of a copy of a computer program and does not allow the sharing of such adaptations.

VI.D.4. Derivative Works and Compilations

Until now, we have been discussing computer programs as if they were a single work, or an original program and a series of derivative works comprising each modification to that program. While that was the case for early computer programs, now it is more common for a program to include preexisting libraries, themselves copyrighted computer programs, and similar components.

When two or more preexisting works are combined to form a new work, in copyright law that work is called a "compilation" – "a work formed by the collection and assembling of preexisting materials or of data that are selected, coordinated, or arranged in such a way that the resulting work as a whole constitutes an original work of authorship." The copyright in the resulting overall computer program comprises the copyrights in the preexisting component computer programs and a new copyright in the compilation. But that compilation copyright is very limited.

The copyright in a compilation or derivative work extends only to the material contributed by the author of such work, as distinguished from the preexisting material employed in the work, and does not imply any exclusive right in the preexisting material. The copyright in such work is independent of, and does not affect or enlarge the scope, duration, ownership, or subsistence of, any copyright protection in the preexisting material.¹⁰⁸

This means that to distribute the overall computer program, there must be permission from the copyright owners of all the component computer programs. It is important before distributing a program using a library that the license that accompanied that library allow the redistribution of the library in the way intended, or else the distribution right for that library will be infringed.

An interesting situation arises when an application program is distributed without the libraries it needs, and those libraries are supplied at a later time by the user of the application program. While this may initially seem like a strange procedure, it actually is common for today's software. Most programs do not run stand-alone on a machine but use the services of an operating system – a special type of preexisting library. Most operating systems also provide dynamic linking to a library, so that the library can be shared by all the programs that are using it.

In this discussion, "library" indicates a preexisting program used by a new "applications program," because that is one of the most common instances of what is being discussed. However, the discussion is just as applicable to libraries that work with a preexisting operating system, or plug-ins that work with a preexisting browser.

¹⁰⁸ 17 U.S.C. §103(b).

With dynamically-linked libraries, the application program being distributed is no longer a compilation that includes the library. Because the library is not being distributed with the application program, no permission is needed from the copyright owner of the library for the distribution to users. Users must, of course, be authorized to use the library, but if they are owners of a copy of the library, under Section 117 they can make any adaptations of the library necessary to use it with the application program.

Some have claimed that an application program that needs a library for its operation is a derivative work of that library. They take that position because the application program is “based on” the library because it was written to use the subroutines and other aspects of the library.

Such a position is misplaced. Even though the definition of a derivative work contained in Section 101 seems to support such a reading when it talks about a derivative work’s being “based upon one or more preexisting works,” the examples all illustrate derivative works where the original work is somehow incorporated or recast in the derivative work:

A “derivative work” is a work based upon one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications which, as a whole, represent an original work of authorship, is a “derivative work”.¹⁰⁹

This need to use a portion of the original work in the derivative work is stated in the legislative history of the Copyright Act of 1976, where the drafters discussed when the derivative work exclusive right is infringed:

To be an infringement the “derivative work” must be “based upon the copyrighted work,” and the definition in section 101 refers to “a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which a work may be recast, transformed, or adapted.” Thus, to constitute a violation of section 106(2), the infringing work must incorporate a portion of the copyrighted work in some form; for example, a detailed commentary on a work or a programmatic musical composition inspired by a novel would not normally constitute infringements under this clause.¹¹⁰

It could be argued that the component program really does include portions of the library that it uses – data structures that are passed as parameters, or even the parameter lists themselves. But elements dictated by external considerations are filtered out when trying to determine whether there is copyright infringement.

No other conclusion makes sense. If it were not the case, then any program using the applications program interfaces (APIs) of an operating system could be considered a derivative work of that operating system. And, under the exclusive right to prepare derivative works, the copyright owner of an operating system such as Microsoft Windows could control who was allowed to write programs for that operating system.

¹⁰⁹ 17 U.S.C. §101.

¹¹⁰ H.R. Rep. No. 94-1476 at 62.