Open Source and Intellectual Property

Presenter: David Elliott

Contents

Introduction

In this talk I want to:

- Review main forms of "intellectual property".
- Review how software development might be (and has been) conceived as a form of intellectual property.
- Consider what "open source" means, and various forms of "open source" compatible devices are in terms of intellectual property.
- Consider whether, apart for standard economic, business, and efficiency arguments, there is any strong moral argument which would favor open source software production over proprietary development.

Caveat: I am NOT a lawyer, nor do I have any legal credentials. I am an amateur viewing an area that is huge in scope, but nevertheless relevant to what I have to talk about. So I have tried to do my best to be accurate, but I do not claim to be absolutely correct about these matters. I therefore welcome and comments or corrections that people might have about my comments about the legal aspects of intellectual property.

Forms of Intellectual Property

The term "intellectual property" is bit misleading. The main forms of intellectual

property--patents and copyright-do **not** permit someone to own ideas; they only permit people to either have a monopoly on their particular expression or manifestation in the form of a useful device or process.

Exception: Trade secrets. This does permit, it seems, the ownership of (particular) ideas indirectly at least. It means simply that certain things cannot be communicated to other people. This is very restrictive, but there are conditions, as we'll see on what can legitimately be a "trade secret".

Main Forms of Intellectual Property (IP)

- Copyrights
- Patents
- Trade Secrets
- Trade-Marks
- Integrated Circuit Topographies

Note: The latter two are not of primary interest to me in this talk.

Note: Canadian Copyright and Patent law is similar to the US, but has differences. (E.g., the "Harvard" Mouse is patentable in the US, but not in Canada. The Canadian Supreme Court has ruled that "higher" life forms cannot be patented. But in the US, apparently higher life forms can be patented.)

Note: An Integrated Circuit Topography is basically a microchip, and the Act provides protection against copying of registered topographies, but does not prevent others from developing integrated circuit products which use *other* topographies to provide the same electronic functions. The point here is that this is mainly a hardware issue.

Traditionally:

1. Copyright law protects literary expression.
2. Patent law protects mechanical inventions.
3. Contract Law protects trade secrets.

In the US, at some point in time, computer software has been protected under ALL three of these forms of IP law.

- **Up to early 60s**: US computer industry relied on trade secret regulations to protect its software.
- **1964**: US Copyright Office began to accept computer programs for registration.
- **1976**: US Copyright Act was passed-but did not explicitly provide for software.
- **1980**: US Copyright Act is amended to include computer programs as "literary works". The US Courts begin to extend copyright protection by (1) considering a program's source code to include the object code, and then later (2) including the logic and sequence of the program as well.
- **1982**: Second Circuit Court of Appeals (New York) judged in *Computer Associates v. Altai* that the basic structure of a program is not copyrightable.
- **1983**: US Patent Office issues a memorandum encouraging software developers to patent their work.

- **1986**: Third Circuit Court of Appeals (New Jersey, Pennsylvania and Delware) in *Whelan v. Jaslow* declared that copyright protection included the basic structure of a program-its lines of written code, and the nonliteral aspects of a program (e.g., the screen design, commands, etc.)-**can be** copyrighted.

Point: Given that all three kinds of IP law have been used-and at least two (copyright and patent) are still in use for software-it is not unreasonable to ask which of these forms of IP law really fit best for software.

What is Computer Software?

Simple definition: A computer program is a procedure for solving a problem-i.e., getting the computer to something in the right way. (Deborah Johnson, "Should Computer Programs Be Owned?" 2000)

This procedure has at least three basic levels:

- **Algorithm**: the solution idea conceived as a sequence of (logical) steps.
- **Source**: the solution expressed in a particular computer language.
- **Object**: the solution compiled into machine language.

Now, if a computer program is to be considered property, it is reasonable to ask which of these features of a program should, and which, if any, should not be regarded as something which could be owned.

Note: I am taking a **philosophical** perspective on this issue; not a strictly factual or legal one. I am asking which body of existing or new law can reasonably be applied to software--not which body actually has been.

Consider now each of the three main forms of IP:

Trade Secrets

Trade secret law standardly applies to "any formula, pattern, device, or compilation of information which is used in one's business and which gives him an opportunity to gain an advantage over competitors who do not use it."

Problems:

1. *All three levels* of computer programs are kept out of the public.
2. Hence this is only useful for internal programming, not software developed for the market.
3. Very difficult to maintain trade secrecy.
4. Prevents others from being able to use the idea to produce other inventions which build on and improve the idea in non-competitive ways.

Copyrights

Copyright confers on the holder the rights to reproduce the copyrighted work, to prepare other works derived from it, and to distribute copies of it to the public by sale or transfer of ownership by lease/rent.

Problems:

1. Traditionally, copyright law forbids the ownership of ideas, but permits

ownership of the *expression* of ideas. (Hence the *algorithm* cannot really be protected-without distortion of the original idea of copyright. If, however, I grasp the underlying structure of a program, I can easily express it in another language.)

2. Generally, it is very difficult to track individual violations; mass produced unauthorized copying is much easier. But, arguably, it is at the individual level where concern about software copying is located.

Also, it is not clear how effective copyright is in a context where:

1. An exact (or near exact) copy of the original expression is possible.
2. The copy can be made for virtually no cost.
3. It can be widely distributed for virtually no cost and very little effort.

This seems true of software, but not "hardware" or other physical, mechanical inventions.

Note: The expression of ideas is perhaps always susceptible to this sort of problem (cf. physical objects). But software seems particularly susceptible to it. This alone does not mean, of course, that it is morally permissible to make illegal copies of copyrighted software. It does, however, raise the moral question about what obligation individual might have to obey a law which is virtually unenforcable.

Patents

A patent is "...any new and useful process, machine, manufacture or composition of matter, or any new and useful improvement thereof..." (US Patent Law).

There are 4 tests for a patent:

Novelty: The invention must be new relative to the "prior art"-all the existing body of technology that could reasonably be known to someone working in the field. This is true for inventions which are essentially the same as some other patented invention OR some other invention widely in use without patent. Also, any invention described in a printed publication or offered for sale more than a year before the date of application for patent is NOT consider to have passed novelty.

Originality: The applicant must be the actual inventor (idea: ensures that a dishonest person cannot take credit for another person's work. Hence patents are granted in the name of a particular person or persons (including corporations).

Utility: The invention must be able to do what its inventor claims it can do. And this purpose or function cannot be illegal or immoral. A perpetual motion machine once put in motion, chemical compounds of no known use, and methods of manufacturing illegal drugs are not allowed.

Non-obviousness: If someone with "ordinary" skill in the area and had access to all existing prior art is likely to consider the invention "obvious," then it cannot be patented. (This test is the on which seems open to the greatest honest disagreement.)

POINT: Patent system can be viewed as a uniform social contract worked out between inventors and society. Both benefit:

- **Inventor**: makes a full disclosure of the workings of the invention and agrees to it entering the public domain after a period of time (e.g., 20 years) in exchange for a monopoly over its use and distribution.
- **Society**: gains by having a useful product which others can learn from and

design other products in ways that are not covered by the original patent. Technological progress is apparently advanced.

Connection between Patents and Copyrights

- **Copyright**: Holder does not have a proprietary right to an idea, only a particular, concrete expression of it.
- **Patent**: Holder does not have a proprietary claim to the useful ideas behind the invention, only to their practical application.

Reason: The rules of IP must not interfere with the freedom of thought and speech. This is (practically) the doctrine/idea of **fair use**.

Open Source v. Proprietary Software

See http://www.opensource.org/docs/definition.php for a full definition.

- *A Simple Definition of Open Source*: Open source software is distributed with access to the source code and some set of restrictions/privileges of use.
- *Proprietary software* is typically distributed typically with only the object code, and hence only the right to use the function under some specified set of conditions.

Note: **Both** can involve a cost-open source does not mean "free", without cost.

Open source software does two things (generally):

1. Provides the holder with a copyright.
2. The copyright stipulates the terms of distribution and use of the software-i.e., sets out the terms of a license.

Different Kinds of Open Source Licenses

1. *Reciprocal Licensing*: The copyright stipulates that any redistribution or significant derivation of the work be distributed under the terms of the original license.
2. *Non-Reciprocal*: redistribution or significant derivation of the work need not be re-distributed under the same terms as the original license.

Examples (1): GPL* and LGPL.*

Examples (2): BSD license. *

GPL (GNU Public License) is fairly restrictive (in this sense) and requires the propigation of the same license for copying and derivation.

Exception: "if identifiable sections [of derivative work] are not derived from the [original software] and can reasonably be considered independent and separate works in themselves, then the GPL and its terms, do not apply to those sections when one distributes them as separate works.

LGPL: the "Lesser GNU Public License" is less strict in that it explicitly allows non-free software (non-GPL or LGPL) to *link to* the LGPL'd software.

Both GPL and LGPL, however, allow one to download, use, modify, share, and

redistribute it-under the condition that the license cannot be altered and the source code must be available. Given these two conditions, it is free for both personal and commercial purposes.

BSD-like, non-reciprocal licenses focus more generally, not only replicating the licensing conditions, but encouraging the wide adoption of the software. Hence far fewer restrictions.

Main Moral Arguments for IP (Property)

- **Deontological:** Property Rights derive from a principle of justice that requires a return for one's labor.
- **Utilitarian**: A person is entitled to own something when this entitlement maximizes good consequences.

Ownership can mean different things, but the sense in which it seems to matter here is the *right to exclude others from use of the program and the right to regulate the buying/selling/renting of others who use it*.

Common Arguments for Open Source

1. Economic/Business
2. Efficiency
3. Ethical/Political

The most interesting moral argument for open source will, I think be one which shows that even if 1 and 2 are not sound, 3 still is. That is, even if (counter-factually, of course, since there are arguably fairly good arguments for each of these claims) the open source model were not optimal in terms of software productivity and did not stand up well in economic/business terms with the proprietary model, it would still be the best ethical choice.

Examples of Moral Arguments

1. *Free Press*: Even if a non-free press were considerably more efficient at distributing information and were more economically efficient, we would probably still prefer a free press over a non-free press. (Richard Stallman has used this example.*)
2. *Selling Babies*: To take an even more stark example, suppose we were to go into the baby "selling" business. We set up baby "farms," hire large numbers of surrogate mothers to reproduce children, and then sell the babies for large sums of money. Suppose this were to have hugely positive effects on the economy, and could easily become an enormously effective business, helping all sorts of people in economically deprived areas, etc. Most people would nevertheless say that this would simply be wrong: We should not be selling-in any sense of this term-persons.

Is there an ethical argument of this nature (although perhaps not as dramatic as the "baby-selling" example!) which might give us reason to support open source development over proprietary development.

Two Steps in a Moral Argument for Open

Source

1. **Negative**: Software is not properly conceived as a form of IP.

   > (The problems of viewing software as a proprietary form of IP have been considered earlier. Additional support for this comes from the rather ridiculous patents that have recently been awarded recently for software. E.g., Amazon's "one-click" web procedure for conducting business transactions. See Tim O'Reilly's discussion for more information.)

2. **Positive**: Software development is best conceived as more like the development of "pure" science rather than "applied".

Claim 2. can be supported by **both** standard ethical approaches mentioned previously:

1. *Deontological*: It is inappropriate in "pure" science (it is against the whole process of knowledge acquisition) to not share results of one's work with others. So to the extent that software production is more like the workings of pure science-i.e., mathematical or logical research, then the proprietary model will equally be morally inappropriate.

2. *Utilitarian*: Sharing results and knowledge in "pure" science is justified, not only because it maximizes the utility of the scientific community, but society's well-being is best promoted by a policy of sharing the knowledge and results of scientific inquiry. Again, if, as I have argued, software development is more like pure rather than applied science, it will equally follow that the open source rather than the (closed) proprietary model will promote social utility.

Note: If it is also true, as is often claimed,. that open source software production is more economical, a better resource for business, and much more socially efficient than proprietary production, then this will only add more to the utilitarian argument just outlined.