Software Licensing

Doug Hockstad

Doug Hockstad is the associate director, software and copyright licensing, at the University of Michigan in Ann Arbor.

Introduction

Software-related technologies comprise an ever-greater fraction of the inventions disclosed by university faculty and staff to technology management offices. Unfortunately, these technologies present challenges to the primarily patent-based intellectual property (IP) policies of most universities. In addition, the unique issues that surface during negotiations involving software often require even experienced technology managers to explore new and innovative approaches to licensing.

Software Protection: Copyright and Patents

Copyright law (the details of which are covered elsewhere in the *Technology Transfer Practice Manual*™) protects "original works of authorship." The types of original works protected by copyright include just about anything written, including plays, music, software, works of art, certain aspects of databases (as described below), and other tangible media. In the United States, the Copyright Act gives the owner of the copyright the exclusive right to (*and to authorize others to*):

- copy the work,
- create derivative works,
- distribute the work, and
- publicly perform or display the work.

How does all this apply to software? Software is, at its core, a written work. In that sense, copyright laws cover the *source code,* or the natural language version of the software. This gives software developers the ability to license their copyrights in the source code to interested parties.

Copyright Registration

Although confusing to many, it is important to understand that you do not need to register a copyright with the government to own a valid copyright. In fact, copyright exists at the moment the creator fixes the work in a tangible medium; that is, as soon as the written work is completed. Whether to actually register copyrights is a decision institutions approach in various ways.

The primary value in registering a copyright pertains to lawsuits: A copyright *must* be registered to bring an infringement suit in federal court. So, it might seem as though owners should always immediately register their copyrights in software (or any covered works).

However, copyright holders can register a copyright at any time...even *after* infringement has occurred. The primary benefit of registering a copyright before any infringement is the ability to seek statutory damages and attorney fees. In industry, this could be a significant consideration. In academia, however, this is not usually a vital consideration.

Another value in early registration is clarity: It marks a specific point in time illustrating copyright ownership, establishing a public record of the copyright claim. Registration can be valuable in any discussions or disagreements over copyright. In the end, if the software is licensed to a company, it might be best to get the licensee's position on registration.

How do you go about registering your copyright? The process is straightforward, if not easy. The process is fully described on the U.S. Copyright Office Web site http://www.copyright.gov and includes filling out one of several registration forms and sending it to the U.S. Copyright Office along with an inexpensive filing fee and some other depository requirements (described in detail on the Web site). Depositing software without appropriate redaction, however, may constitute a "public" disclosure of the code that could harm the potential for patent rights or allow others to create unauthorized copies of the code.

Patents

Software authors will often request that the licensing office "patent" software. Owners of inventions will often seek patent protection for software through patents covering the process embodied in software. The Court of Appeals for the Federal Circuit recently clarified the standard for whether processes (such as those found in software) are patent eligible. The court in *In re Bilski* ruled that a process is patent eligible if it either (i) is tied to a particular machine or apparatus or (ii) transforms a particular article into a different state or thing.

The U.S. Patent and Trademark Office (PTO) and courts need to further interpret the practical meaning of this standard that will apply to most patents aimed at protecting software. In the meantime, before filing a patent on any software, it is worth discussing with your patent prosecution counsel whether the invention embodied in the software not only has commercial value, but also whether it can be claimed in a patent in compliance with the new *Bilski* standard.

In many, if not most, circumstances, the value of a particular software package lies in (a) the fact that someone took the time to create it and (b) how hard it would be to re-create. In general, if time and money were no object, a person could write his or her own version of most software programs that otherwise would be purchased.

Why doesn't this happen more often? Usually not because of patent protection (although in some cases this might be a deterrent), but more often because the software is priced lower than the value of the time and resources required to re-create it. It is simply more cost and schedule effective to purchase the product than to re-create it. In addition, it is often more cost effective to use established software product maintenance, based on product support of multiple customers. Accordingly, quite often, a university can adequately protect its software inventions through copyright and does not need to pursue costly patent protection to effectively commercialize the software.

Patents on software *do* have value in certain situations, however. In some cases, the software itself is trivial to write, but what the software does, and how it does it, is so unique

and unforeseen, that a patent is the best way to capture that value. In other words, the value of your technology may lie more in the novel function the software performs, rather than in the source code itself. In these cases, when there is something unique about what or how the software works, a patent may be the best choice, even though the time and expense of getting a patent is considerably greater than that of establishing the copyright.

Another consideration about patenting software: enforceability. Source code is often invisible to the user. Therefore, if you cannot easily detect whether a third party is practicing your new process, then your patent will have little value to your university or a licensee, and filing for a patent is likely not worth the cost.

Finally, you should also consider the foreseeable life of the software when deciding the best IP protection to pursue. Patents, in general, take a relatively long time to obtain, and software patents take even longer (currently it takes approximately four years simply to receive the first PTO office action on software-related applications). Accordingly, licensing staff should consider whether patent protection might be definitively obtained during the period of time when the software still has commercial value in the market. If not, copyright protection (which, as discussed above, is immediately effective upon writing the software) is likely the more effective method of protection.

Databases

Databases, while not necessarily software, are usually embodied in software and, therefore, such invention disclosures often fall under the responsibility of the software licensing professional. In actuality, databases are one (or both) of two items: an organization methodology (the structure of the database) and the data (facts) stored within the database. Under this framework, the data themselves are not protected by copyright laws.

However, the organization of the data, the uniqueness and layout of the database structure, is potentially protected by copyright. There must be some original authorship reflected in the organization and layout of a database. While the alphabetical listing of names in a phonebook does not constitute original authorship, a database that categorizes or relates data in a thoughtful manner may contain a copyrighted organization or

layout. In that case, it is important to remember that copyright will only protect that particular layout, and not the underlying data.

While difficult to protect by copyright, databases can be protected in other ways. Most often they are protected and exploited simply through contract law. That is, access to, and use of, the database (whether meaning the structure for storage or the data contained within) is controlled by a usage license, restricting what the licensee can do. These licenses can be extremely valuable, both in terms of revenue and simply institutional and author acknowledgment, which is often an under-appreciated benefit to universities.

When licensing data, it is important to ascertain and verify the sources of the data. Data are often not just "created" by faculty, but gathered and collated from a variety of sources. Many of these sources might have contractual restrictions on use, or even restrictions on data derived from its use! Individual faculty members will often not be aware of such restrictions, and, therefore, you run the risk of licensing software containing third-party IP, exposing the university to legal liability.

Ownership and IP Policy

Scholarly Work

Since Bayh-Dole, and at some universities even prior to Bayh-Dole, IP policies attempt to address the typical situations in which the university would claim ownership of IP and how it might protect and commercialize such IP. Many policies describe these ownership principles in terms of patentable technology. They do not anticipate other forms of IP protection that may be better for different types of technologies that were potentially unforeseen at the time the institution drafted the policy (such as software).

In addition, there may be confusing overlap between university copyright policies drafted long ago to address faculty ownership of scholarly work (such as books and curriculums) and more recent IP policies (aimed at technology transfer or patent issues). Absent any direction from the institution's IP policy, copyrighted works, and software in particular, present some unique opportunities for confusion and misunderstanding that licensing offices are left to address on a case-by-case manner.

One of the most common misunderstandings is related to what is commonly referred to as *scholarly work exceptions.* In industry, employment agreements usually clearly state that employers own any employee creations related to their work, consistent with work-made-for-hire requirements in copyright law. Universities often have similar agreements, but almost always also have scholarly work exceptions allowing faculty to retain ownership of such work.

The term *scholarly work* has the trouble of being, relatively speaking, undefined. Historically, the term covered such things as journal articles, books, works of art (e.g., music, paintings, sculptures, etc.) and instructional materials (unless, of course, any of these materials were the specific results of either a funding source or a specific assigned duty) created by faculty at the university. The problem is that, often, this scholarly work exception is not clearly defined, and various software works could be interpreted to fall into the definition of scholarly works. In many cases, universities have been slow in clarifying the issue, often leaving their faculty (and the technology transfer office) in undefined territory.

As an illustration, consider the following situations at hypothetical University X. For these examples, University X has an IP policy stipulating the university owns copyrighted material outside the definition of scholarly work as long as the material is either related to the employee's position at University X or involved use of University X facilities. Under this policy, if a physician writes software related to medicine, likely he or she used some amount of university resources to accomplish this and/or used his or her experiences at the university in developing the software. Accordingly, the likely result is that University X owns such software.

Similarly, if a radiologist writes a program to manipulate digital medical records, likely University X also owns that software. If, on the other hand, the same radiologist publishes a journal article describing a new system for manipulating digital medical records, then the radiologist would likely maintain ownership of that article as a traditional scholarly work.

In either situation, regardless of who owns the software or journal article, if either were created as a deliverable under a government grant, the university is responsible for compliance with reporting requirements and ensuring the government receives a retained government use license in such software or article.

Alternately, if an emergency room physician writes a new golf simulation program, University X might not own that program because it falls outside of the physician's employment and experiences at University X.

Because of this lack of clarity, disputes may arise and should simply be worked out on a case-by-case basis. Faculty members often feel strongly that the software they create should be their own. In addition, it often takes lengthy conversations and review of the funding sources to determine who really *does* own or have rights to such software. The best course of action is to review and, if needed, revise an institution's IP policy to clarify software ownership.

Revenue Sharing

By policy, custom, and, ultimately, the Bayh-Dole Act, universities share licensing revenues with inventors (in stark contrast to most company policies). This can make inventorship (authorship) a contentious issue because identifying the inventors (authors) can be difficult. By law, the author of a copyright work is the person who created it (or his or her employer if this is work made for hire), not the person with the idea or the person who led the project. Such problems often lead to confusion over who should be listed as the inventors of software.

Most universities take one of two positions. The first is that revenues related to software will be shared with the idea originators and any others who materially contributed to the concept and development of the software...except for the programmers. In this model, the programmers are being paid to program, offering little in the way of intellectual content and, therefore, not eligible for royalty sharing.

The second model includes programmers as *contributors* to the software invention who are eligible for royalty distribution. Most institutions appear to be converting to a model that includes programmers as contributors. The reasoning behind this is two-fold: Most often the other authors subjectively believe that the programmers significantly contributed to the software invention and should share in any proceeds; and in most cases, objectively speaking, it would be hard to envision a case where the programmer did not significantly contribute to the software invention. This appeals to both a sense of fairness and practicality.

Software Development

Software is developed under many scenarios, but most often either as a result of someone needing specific tools he or she cannot find commercially (or can't afford) or as a specific planned outcome of a funded project. Regardless of the circumstances of development, software developers usually possess at least one of three desires concerning financial remuneration associated with their software:

- for revenues to support ongoing development of the software's parent project
- to provide the software for free (most often seen when software is part of a collaborative effort with others)
- to personally profit

License Revenue-Supported Projects

Commonly, academic software developers desire to support further development or support of the project through license revenues. While laudable, there is often a reason to discourage this: If there are authors (or inventors) associated with the project, there will be tax implications to waiving their revenue share to an account in the university that they control. In most situations, this is considered first as revenue they have earned (taxable) and then as a donation to the university (tax deductible)—the two transactions do not simply cancel out, and the personal liability of the authors remains. Many institutions avoid this problem by disallowing the direction of waived revenue: Such revenue goes to the department and is not in the control of the inventors. It is then up to the department to decide whether or not to fund the project from such revenues.

Third-Party Software

Finally, note that much of the software disclosed to university technology transfer offices will contain third-party components. These are the utilities, libraries, functions, and more that programmers find on the Internet, or purchase, to accomplish a specific task within their own work. For instance, someone writing a word processor might use a publicly available spell-checker rather than create one from scratch, saving substantial time and effort.

Usually, there is no malicious intent in including the third-party component code. Programmers often believe the code is free: free to use, free to distribute, free to do with whatever they please. In many, if not most, cases, this is *not* true. There is almost always a license associated with the use of the component, and it is the responsibility of the technology manager to carefully assess these licenses and determine the best course of action.

In some cases, it is permissible to include the software in the end product for certain purposes (e.g., open source or academic purpose versus commercial use). In other cases, the component should be replaced with another having a more flexible license or be completely rewritten from scratch. All too often, academic software is released without removing these hidden traps, much to the consternation of the licensing reps as well as the general counsel's office.

A Special Note on Open Source in Development

Open source software continues to become more prevalent in the university setting, presenting significant issues when faculty or staff incorporate it into their software. While *open source* is a term used differently from situation to situation, it commonly refers to software licensed under specific terms meant to encourage its open use. Some of these open source terms include unrestricted redistribution, access to source code, and rights to derivative works.

Some examples of commonly referenced open source licenses include the GNU General Public License (GPL), the Berkeley Software Distribution (BSD), the MIT License, and

the Mozilla Public License. If a faculty member has used any open source software during the course of developing his or her software, it will be important to analyze the terms of the relevant license to determine whether the faculty member's own software must be released under the same license as well. While open source licenses share many common principles, they often differ in important ways and should be analyzed individually. (For more on open source licenses, see "Open Source Software Licensing," in Volume 3 of the *AUTM Technology Transfer Practice Manual, 3rd Edition.*)

Software Licensing

While similar to any license from a university, software license agreements should contain some unique attributes. One unique aspect of software licensing is the wide potential range of commercialization paths, including third-party development firms, end users, and distributors. As addressed below, each of these potential commercialization paths has pros and cons, and the nature of the particular software to be licensed will suggest the appropriate licensing path to pursue. In particular, some points to consider in selecting one path versus another include:

- How robust is the software?
- Will it require support?
- How much ongoing development is planned?

Which licensing path you elect to pursue (through in-depth discussion with the involved faculty) will drive the particular form of the license and the important terms to consider.

One potential commercialization path is a software development firm. The majority of software developed in the university setting is not of commercial quality. That is, the software has not gone through extensive debugging, optimization, and quality-control phases, or does not have extensive documentation. This possibly makes the best commercialization path a third-party software development firm, which would put the software through a rigorous release process. In the end, this path provides a better product to the market. Such license agreements are similar to traditional university patent agreements, providing the licensee the right to create its own works based on the university's work (e.g., derivative works), requirement to indemnify, no warranty clauses, etc.

Another potential commercialization path is direct to end-user. This path is not typically available to early-stage university technologies. Software that is already successfully used internally, and/or has high ease-of-use, might be a good candidate for such licensing, as might software with limited appeal outside of small markets. Such licenses are typically shorter in length than traditional university licenses, not requiring much of the reporting and diligence sections.

However, these agreements should contain the same warranty disclaimer clauses. In addition, care should be taken that the university is dealing with qualified licensees (whether from a financial perspective or from an export controls perspective), and that individuals within an organization are not able to inappropriately accept a license on behalf of their company (e.g., there should be language that covers either the individual or acceptance on behalf of an organization). In some cases, extra care should be exercised when the software has the potential to do harm to either the licensee or the licensee's equipment.

Distributors are yet another commercialization path for software, where the licensee simply acts as a sales agent for university software. Although software following this path may have already achieved a relatively high-quality state of development, similar to the end-user example, this software might have a steep learning curve, or be prone to significant support requirements, and likely requiring more resources (e.g., ongoing support and consulting) than a university could, or is willing to, supply.

A reseller or other distributor would better serve this type of software market. Distributor/reseller agreements are similar to the traditional developer agreements above; but these licenses would not include the right to create derivative works. Instead, the licensee's business model is one of margin on sales and support fees.

Open Source Licensing

There is a large movement by universities and other software developers to support distribution of free software. Free can mean anything from free to make derivatives, to no cost, or many things between. As noted above, faculty creating software are often driven to create solutions for the public good and have little or no desire to obtain personal

profit. Further, if the software was created as part of a collaborative effort with others, there may be expectations or requirements to contribute such software back for use by others. (Note, there is not a statutory equivalent to Bayh-Dole for copyright.)

In these cases, it is the technology manager's responsibility to (a) ensure faculty and staff understand the value of the material they have, (b) understand the consequences of their decision to have the software provided for free, (c) protect the institution from harm, (d) and ensure appropriate institutional or author acknowledgement or credit. Also, it's important to explain the different types of free, or open-source, licenses available.

Prior to using any specific free-use, or open source, agreement, it is important to discuss with the creators their overall objectives for the software as well as the consequences of this choice. The official open source definition by the Open Source Initiative is found at http://opensource.org/docs/osd. In essence, open source software meeting this standard has the following attributes (among others):

1. Freely re-distributable
2. Source code available
3. Ability to create derivatives
4. No discrimination against any user group or use

A commonly referenced open source license is the GPL (GNU General Public License). Often referred to as *copy-left*, this type of license is viral in that it requires any software that uses, incorporates, or is derived from software licensed under it be released under the same license. [Note: The GPL is currently undergoing a significant revision (version 3.0) and should be reviewed by counsel to catch differences from the prior, well-known version 2.0.]

However, free software doesn't have to be released under the GPL. There are many styles of free use and open source software licenses. The most important thing is to clearly understand what the goals of the faculty are:

- Do they simply want it to be free of charge?
- Do they want to distribute executable code or source code?

- Do they want to allow derivative works?
- Do they want to allow licensees to distribute it further?
- Do they want licensee derivative works *back*?
- What is their real goal in making this open source?

Technology managers should discuss all of these issues carefully with faculty prior to releasing software under any open source (or free) license.

A Final Note...Click-Wrap Licensing

Software and other digital media are often protected by what are called *click-wrap licenses.* Such licenses are typically for use as end-user licenses and allow licensees to simply accept a license electronically rather than in written form, with no negotiation of the terms. Click-wrap licenses are presented to the user at various points, usually while downloading the application, installing it, upon first running it, or in the case of online databases, upon first accessing it.

Generally, the user must affirmatively consent to the terms of the license agreement (the entire license presented, with the user required to scroll through it) to become bound by it. These agreements are a useful tool for the licensing professional due to their inherent reduction in negotiations required for each licensee. However, consultation with attorneys is recommended prior to using them, as they have not been diligently tested in court yet, and some software might be more risky than other software.