12

Open Source Litigation

Owning a Cause of Action

The prospect of litigation over open source software is disturbing to all of us. Open source software cannot flourish in a litigious environment in which everyone is suing everybody else over perceived injustices relating to open source licenses.

Indeed, in practice, there is very little litigation over open source. After all, why would a licensor who is permitting everyone to copy, modify, and distribute his or her software need to complain about someone who did those things? And why would a licensee who receives software with essentially unlimited rights to it need to demand even more from the licensor? When the software is essentially *free* (i.e., zero price), and when software freedom is guaranteed by the license, why would anyone bother to sue?

But litigation can occur, and it is important for anyone involved with open source software to understand the risks.

The risks are low. If you honor the terms of the licenses for open source software you acquire, you probably won't be bothered. And whatever litigation risks you do accept with open source software are essentially the same risks as with proprietary software. If you live in a litigious society, you need to be prepared for lawsuits.

A *cause of action* is simply a matter for which a legal action may be maintained. In the open source context, causes of action can arise over intellectual property matters, such as ownership of copyrights or patents, and interpretation or enforceability of license and contract terms, and for business practices that are perceived by one party or another to be unfair. A cause of action is said to be *owned* by the party that has the right to maintain it in court.

When a licensee accepts software under an open source license, he or she acquires nonexclusive rights to intellectual property in the software, including the rights to make copies; to create and distribute derivative works; and to execute licenses to make, use, and sell products containing that software. The licensor, you will recall, has made promises (express or implied) to each licensee concerning the availability and quality of the software. A licensee may sue in court to enforce those promises, even if it means suing the licensor who gave him or her that software in the first place or suing third parties who improperly interfere with the practice of those rights. A licensee, then, can potentially own one or more causes of action and be the plaintiff in a lawsuit.

A licensor distributes software under an open source license containing certain terms and conditions that licensees must obey. Licensors may sue their licensees in court to enforce the terms and conditions of the license or to terminate it. A licensor, then, can potentially own one or more causes of action and be the plaintiff in a lawsuit.

A contributor participates in an open source project and submits his or her original works of authorship to the project. The contributor may sue to protect his or her copyrights and patents from those who use that software outside the scope of the license (express or implied) to the project. A contributor,

then, can potentially own one or more causes of action and be the plaintiff in a lawsuit.

A stranger may own a copyright or patent that is embodied in open source software without the stranger's authorization. He or she may sue to have that infringing intellectual property removed from the software. A third party, then, can potentially own one or more causes of action and be the plaintiff in a lawsuit.

Finally, there are societal interests at stake in open source software. Governments may promulgate software export laws, mandate standards for security, and enforce antitrust rules. Bankruptcy laws may interfere with ownership of intellectual property. These interests may be enforced in court, sometimes directly by the government. Governments, or government agencies, can potentially own one or more causes of action and be plaintiffs in lawsuits.

Owning a cause of action, of course, doesn't necessarily mean that you will win in court. All you have is a right to institute judicial proceedings, and it will be the judge or jury that will decide, based on the facts and the law, whether the plaintiff or the defendant wins.

Damages

The main reason we worry about litigation is because of the consequences of losing. The other big reason is the cost of the litigation itself. For major battles between big companies, attorneys' fees of more than $300,000 per month are now commonplace in the United States. Ignoring attorneys' fees for the moment, though, what are the potential consequences of losing a lawsuit?

Calculating damages arising from cause of action in a software dispute is tricky. What is the value of software? Is it a

function of the price paid for the software or the benefit derived from the software? Are damages a function of what was lost, such as business opportunity or sales? If the damages were caused by a part of the software but not the entire package, should damages be prorated?

There are no default damage calculation rules, although some licenses vaguely address this problem (e.g., MPL section 8.3; OSL/AFL section 11). There are also no standard royalty rates for copyrights or patents against which damages can be calculated.

The prospect of damages may encourage a company to file a lawsuit, but it probably shouldn't unless there is a reasonable prospect of recovering at least enough in damages to pay for its own attorneys' fees and costs.

I once represented a company that wanted to sue because a licensee hadn't complied with a provision of the GPL that requires the licensee to give recipients of the Program "a copy of this License along with the Program." (See GPL section 1.) While that was technically a violation of an express GPL condition, how should one calculate damages for its breach? How much would my client have to pay his own attorneys to force the licensee to either obey the GPL or pay damages for infringement? And then, how should a court calculate damages for the failure to publish a license that anyone can find instantly on the Internet? Our final problem was that, by the time we had discovered the licensee's failure to publish the GPL as required, the licensee had already stopped distributing his software. How can we calculate damages for *past* breaches of a license that are not ongoing?

Perhaps unfortunately for those who would welcome the clarity of a court decision, such questions were never answered because my client decided not to sue. No court has yet told us how to calculate damages for breaches of open source licenses.

Answers to these questions will depend upon the specific business and software facts of the case and upon local law.

All open source licenses—indeed, all software licenses of any kind from commercial companies—contain limitations of liability. This is to ensure that the maximum dollar exposure of a party for damages due to claims by the other party is strictly limited. (In some jurisdictions, class action lawsuits can aggregate the small damages of a large number of plaintiffs into one large claim on behalf of all members of the class; this possibility is well beyond the scope of this book.) As for the maximum dollar exposure for such claims, all open source licenses essentially contain provisions that say "no damages at all."

Limitation of liability provisions are not enforceable in all jurisdictions, despite what the license says. In some countries, consumer protection policies always trump a vendor liability disclaimer.

The limitation of liability provisions in the BSD, MIT, Apache, GPL and OSL/AFL licenses protect only the licensor; in the MPL and CPL, they protect both parties. Some limitation of liability provisions purport to limit liability to any person; see MPL section 6. It is difficult to see how such a limitation in a license between two parties would be binding on a third party.

So even where damages can be calculated, the limitation of liability provision may reduce the actual recovery.

Where actual damages are difficult to calculate, statutory damages may be prescribed by law. Statutory damages for copyright infringement in the United States can range from $750 to $30,000 "as the court considers just," and in cases of willful infringement the maximum statutory damages are increased to $150,000. Damages are calculated for the entire work and not for each copy made:

> *...For all infringements involved in the action, with respect to any one work, for which any one infringer is liable individually, or for which any two or more infringers are liable jointly and severally.... For the purposes of this subsection, all the parts of a compilation or derivative work constitute one work. (17 U.S.C. § 504.)*

The prospect of collecting statutory damages often isn't enough to compensate for attorneys' fees and costs. For example, in the case I described earlier where a licensee had merely failed to publish the license as required by GPL section 1, an award of more than the minimum statutory damages of $750 is unlikely. After all, why would a court consider higher amounts just under the circumstances?

Nor should a prospective litigant rely on a provision of a license or of a statute that awards attorneys' fees to the prevailing party. Such awards are often limited to "reasonable" attorneys' fees, and they may also be at the discretion of the court.

> *In any civil action under this title, the court in its discretion may allow the recovery of full costs by or against any party.... Except as otherwise provided by this title, the court may also award a reasonable attorney's fee to the prevailing party as part of the costs. (17 U.S.C. § 505.)*

Injunctions

Usually an injunction is of far greater concern to a defendant than monetary damages. An injunction is:

> *A court order prohibiting someone from doing some specified act or commanding someone to undo some wrong or injury. (Black's Law Dictionary, 6th edition.)*

Injunctions will be ordered by a court when economic damages are not adequate to compensate for the wrong. On the other hand, courts are reluctant to issue injunctions when monetary damages would be sufficient to redress the wrong.

Consider the financial repercussions to a company of being ordered by a court to stop using software that has become an essential component of that company's processes or products. Risks like these often make injunctions far more frightening than monetary damages.

In the previous section I described a situation in which a licensee had failed to publish a copy of the GPL with his software, in violation of GPL section 1. My client realized we might not recover much in damages, but at least we might be able to encourage a court to grant an injunction against any further use by that licensee of my client's software.

But would the court find that this was a "material condition" of the GPL whose breach could justify such a dramatic remedy as injunction? Such questions are particularly troublesome for bare licenses like the GPL, because the concept of materiality of a condition is found only in contract law. One would hope that courts would balance the equities in such situations so as to avoid terminating open source licenses for simple breaches that can easily be cured (i.e., by simply publishing the license).

On the other hand, the threat of an injunction can often cause licensees in breach to cure their breaches before the court acts.

In my client's situation, unfortunately, the licensee had already stopped using that GPL-licensed software, so an injunction was moot anyway. We ultimately never tested any of our damages or injunction theories in court.

Standing to Sue

Not everyone who perceives a wrong is allowed to sue to correct that wrong. Parties to litigation must have a suitable stake—a legally protectable and tangible interest—in the outcome of a dispute. *Standing* to sue deals only with the question of whether the litigant is the proper party to fight the lawsuit, not whether the issue itself is justiciable.

Open source licenses often elicit passionate support in the open source community. That passion does not necessarily translate, under the law, to *standing*. Only parties with a well-defined legal interest in the outcome may litigate an open source license. Even open source advocacy groups such as the Free Software Foundation and Open Source Initiative don't have standing to sue to protect software freedom or to protect software under open source licenses. Nor is the public an *intended beneficiary* of open source licenses, despite the open source goal to serve the public interest in software freedom. A mere member of the public can't sue to enforce an open source license.

Intellectual property laws narrowly limit standing. Only the owner of a copyright or patent may sue to enforce the copyright or patent. Distributors who don't own copyrights or patents can't sue under copyright or patent law to enforce their contributors' copyrights and patents, but they do have standing to enforce the copyrights and patents embodied in their own collective or derivative works.

Since the GPL is intended by its authors to be a copyright license but not a contract, and since there is usually no attempt to seek assent by licensees to the terms of the GPL, that license presumably cannot be enforced under contract law. All the other licenses described in this book are designed to be contracts and so the parties to those licenses can sue to

enforce them as contracts. The parties to a contract have standing under contract law to enforce that contract. This means that licensors and licensees can enforce their licenses that are contracts, regardless of who owns the underlying copyrights or patents.

Burden of Proof

Consider first what would happen in a typical licensing dispute under copyright law for a bare license. (Refer to the comparison of bare licenses and contracts in Chapter 2.) A plaintiff will allege that the defendant is a copyright infringer and thus may not exercise any of the exclusive rights of the copyright owner.

1. The plaintiff will have to prove he or she is indeed the copyright owner. Only the copyright owner (or, in the United States, an exclusive licensee) has standing to sue to enforce the copyright.
2. The plaintiff has the initial burden of demonstrating that the defendant has undertaken one or more of the copyright owner's exclusive rights under the copyright law (e.g., made copies, created derivative works, or distributed). The defendant, as always, can defend him- or herself on this issue (i.e., not everything is a derivative work simply because a plaintiff calls it that; see the discussion of derivative works analysis later in this chapter).
3. The defendant can assert the license as a defense to infringement. In essence, the defen-

dant can admit to making the copy or creating the derivative work, but assert that the license authorizes this action. (If the defendant admits to the infringing acts but denies the existence of the license, of course, the defendant is an infringer.)

4. The plaintiff may then prove that the defendant breached a condition of the license, thus rendering it terminated or revoked. The conditions of the license will be interpreted by the court under local law standards as appropriate for bare licenses.
5. The plaintiff bears the burden of justifying injunctive relief and proving damages.

Notice that in a copyright dispute over a bare license, the plaintiff will almost certainly be the copyright owner. If a licensee were foolish enough to sue to enforce the terms and conditions of the license, the licensor can simply revoke the bare license, thus ending the dispute. Remember that a bare license in the absence of an interest is revocable.

It may be that bare licenses will be interpreted by the courts under contract law principles, even in the absence of the contract formalities of offer, acceptance, and consideration. After all, major software companies around the world distribute open source software as part of their products; those open source licenses may be technically and economically impossible to revoke. Furthermore, in commercial dealings of any significance worthy of being turned into litigation, there are almost certainly other aspects of *offer, acceptance,* and *consideration* that can be invoked by creative lawyers as proof that a contract was formed.

There are some important differences to this scenario if this becomes a contract dispute, where the license has been offered and accepted, and consideration has been paid. Now not only does the licensor have standing to be a plaintiff regardless of whether he or she owns the copyrights and patents, but also the licensee has standing to be a plaintiff to enforce the terms of the license and to prevent it from being revoked. The statutory and case law of contracts (at least in the United States) would guide the court to interpret the license and to determine whether there was breach of contract and, if so, what damages or injunctive relief should be granted.

The remedies of copyright and patent law are fairly broad-brush. The defendant is either an infringer or not, and must either obey the terms of the license or see it revoked. Damages are to be awarded as specified in the relevant copyright or patent statute.

Contract remedies can be more nuanced, however, and they may become very effective for open source license disputes. For example, one of the more interesting remedies available for contracts—but not for bare licenses—is "specific performance," by which the party breaching the contract may be ordered by the court to perform. Specific performance is not a remedy for a dispute over a bare license.

At the end of the day, the parties to an infringement dispute in court will often finally resolve it by drafting their own settlement agreement that allows the intellectual property to be used. Even if there was no contractual license initially, that settlement agreement will be a contract and license that is enforceable in court.

How much cheaper it would be to draft a good open source license up front, get the parties to agree to it as a contract, and proceed upon those agreed terms.

Enforcing the Terms of a Contract

Proving breach of contract has been the subject of literally millions of lawsuits. It would be impossible to summarize that body of case law and statutes effectively in this book. Indeed, contract enforcement depends in some ways on the jurisdiction in which the case is brought, and most such cases are fact-specific. I will list only the major rules that apply in many jurisdictions:

- Courts will generally try to give effect to the written contract of the parties. Parties are allowed to agree to almost anything as long as it is not against public policy.

- Aggrieved litigants are not allowed to back out of contracts they made simply because the terms are no longer to their liking. It usually doesn't generate sympathy if you complain after the fact that a contract you entered with your eyes open is now unfair.

- There are complex rules for resolving ambiguities of contract language, and the courts will often try to reword such ambiguities to make the contract enforceable. If the ambiguity is so profound that the parties probably didn't understand what they were agreeing to, the entire contract may become void. (In the absence of a contract, remember, copyright and patent laws remain in effect; a party who acts under authority of a void license is merely an infringer.)

- There are complex rules for filling gaps in contracts where the agreement is silent as to specific matters. Commercial relationships among countries have led to the development of consistent laws relating to the sale of goods. Whether software is goods depends on the laws in your jurisdiction. In many cases, though, courts will make an analogy between software licenses and contracts for the sale of goods, thereby developing case law where statutory law about software isn't complete.

- Contract terminology that is not defined will often be given its meaning as a term of art. In complex cases, courts may rely on expert witnesses to help them determine the effect of specific contract language. Among the terms of art relevant to software licenses are *collective work*, *derivative work*, *copy*, *distribution, file,* and *module*. Courts will apply case law and statutory interpretation processes to determine the meanings of such terms and their effects on specific licenses and software.

- Commercial parties are generally assumed to be sophisticated about the contracts they enter; they will find it difficult to argue that they didn't really know what they were agreeing to. Individual consumers, on the other hand, are not sophisticated; they probably didn't even read or understand the consequences of software licen-

ses they "agreed" to. Courts may protect individual consumers from unfair license conditions where they wouldn't bother to protect a sophisticated company whose lawyers reviewed (or should have reviewed) the licenses.

- Courts sometimes refuse to enforce specific provisions of contracts against ordinary consumers, particularly if those provisions are excessive burdens on unsophisticated licensees. For example, arbitration clauses, broad warranty and liability disclaimers, and biased selection of jurisdiction, venue, and governing law may not be enforced against naive licensees. No court case has yet tested whether a reciprocity provision can be asserted against an unsophisticated licensee, although big software companies can be presumed to know what those provisions mean.

I recognize that these guidelines don't provide much real guidance for anyone who is considering suing for breach of contract or who fears having to defend against such a lawsuit. Fortunately, the open source community is not particularly litigious. Licensors give away so many copyright and patent rights that there's very little left of value worth suing over. And licensees obtain almost everything they need to profit from the software, so there's very little incentive to sue. Without damages, lawsuits aren't needed.

Nevertheless, licensees should be diligent in respecting the intellectual property rights of contributors. Honor all the terms and conditions. Little things often matter deeply to open source licensors. For example, if a license requires that you make available a copy of the license or of the source code

when you distribute the software or its derivative works, do so. The open source community generally believes that such license terms are really worth fighting over, so avoid such fights by obeying the license terms and conditions.

Disputes over Ownership of Intellectual Property

Licensors can only license software which they own or which they have received permission to license. That basic legal requirement is explicitly acknowledged in the OSL/AFL by the warranty of provenance and in the MPL and CPL by their representations. (OSL/AFL section 7; MPL section 3.4[c]; CPL section 2[d].) All open source licenses, regardless of their explicit language, at least imply that the software is being licensed under the authority of its copyright owner. A licensor who fails to abide by that implied or explicit promise can be guilty in some jurisdictions of fraud or gross negligence, regardless of warranty disclaimers.

A contributor who submits a contribution he or she doesn't own might be forced to pay damages to cover the cost to replace the infringing contribution or to buy a valid license from its rightful owner.

Companies that make contributions to open source projects are assumed to be sophisticated enough to take responsibility for the software they contribute. But sometimes employees make contributions that their employers do not approve or allow. That is really a dispute between the employee and his or her employer. Recipients of such unauthorized contributions may allege negligent supervision if employers fail to supervise properly their employees' participation in open source development.

This means that companies that participate in open source development should document their procedures and policies

to their employees. Attorneys should review those procedures and policies to protect companies' intellectual property.

Recipients of open source software under apparently valid licenses may suddenly find their software challenged by third parties claiming ownership rights. This is in part what happened in the SCO *vs*. IBM litigation, where SCO claimed that IBM had no authority to license certain software under the GPL, software that ended up in Linux. Open source is not unique in this respect; such ownership disputes can also occur with proprietary software. Licensees are not direct parties to those intellectual property ownership disputes, although their licenses might ultimately be affected by the outcome.

There is little that downstream licensees can do in advance to avoid third party claims to intellectual property against their licensors. Some licensors are now offering to indemnify their customers against such claims, although any indemnification paid will often be worth far less than the infringing software those customers can no longer use.

When third parties prove their valid claims to ownership of open source software, only one response is appropriate: The software may no longer be used without a license from the true owner. Open source licensing depends on intellectual property law, and it would be hypocritical of open source distributors and customers to dishonor those laws by copying software to which they no longer have a license.

Disputes over Derivative Works

I left for last the most difficult legal question facing the open source software industry: What is a "derivative work" of software?

If an open source license doesn't have a reciprocity condition, derivative works simply don't cause problems. You can

safely ignore this topic entirely if you license software under an academic open source license.

Early in this book I explained the complex problem of separating expressions from ideas, art from science, and right brain from left brain creations. To determine whether a software program is a derivative work of another software program, the courts need to disentangle these abstractions. The procedure that many courts use, called the *abstraction-filtration-comparison* test, is described below.

I previously took the easy way out. I said that you should treat derivative works as subsequent versions of an earlier work. But that easy way out no long suffices; works resemble each other in many subtle ways. For example, Microsoft Excel 2002 is probably a derivative work of Microsoft Excel 2000, but is it a derivative work of Lotus 1-2-3? Of Visicalc? Is Linux a derivative work of UNIX? Is the implementation of software conforming to an industry standard a derivative work of that specification? How much copying of source code is required to create a derivative work? How much copying of source code may you legitimately do before you create an infringing derivative work? Does linking create a derivative work?

These questions are important to some licensees because they want to avoid the reciprocity conditions of open source licenses, and they are important to licensors because they want to enforce those reciprocity conditions. Disputes over whether particular software is a derivative work of licensed software, and thus subject to reciprocity, are inevitable.

A derivative work, you will recall, is a work based upon a preexisting work. The preexisting work is modified, translated, recast, transformed, or adapted so as to create an improved (or at least different) derivative work. (17 U.S.C. § 101.)

In theory, different copyrightable works, including software, can be compared to determine whether one is a deriva-

tive work of the other. This may involve a comparison of the source code or the object code, depending upon the facts of the specific case.

Expert assistance may be needed. We may have to perform reverse translation or automated source code comparisons to identify similarities between two programs for presentation to a court. If we only have object code, we may have to compare object code versions or reverse-compile the software to create easy-to-read versions. This first step is itself complicated, because the parties to the dispute have to reduce the software similarities to simple constructs that can be recognized by nontechnical judges and juries.

In the simple case, two programs can be set side by side and their source code compared. A program that is substantially similar to a preexisting program is likely to be a derivative work. That is because such similarities rarely occur by coincidence, at least for substantial portions of the source code. But *substantial similarity* (a term of art in copyright litigation) is not enough to identify a derivative work.

Some similarities relating to the basic functioning of computer systems (e.g., subroutine entry and exit code, external interfaces) can occur by coincidence or intentionally because "that's the way computers have to work." Some snippets of software may be too small and ordinary to be copyrightable. In other cases program functions are coded in a particular way because that is the only (or most effective, or the industry standard) way to implement that specific function on that particular computer architecture. Such source code must be excluded from the comparison because it is not entitled to copyright protection; instead, it is idea that has merged into expression, and is thereby rendered uncopyrightable.

In practice, comparing two works of software is not as simple as a byte-by-byte or line-by-line scan. Software is often

extensively modified between versions. Entirely new coding techniques, programming languages, and interface designs can make software appear to be very different at the source code level even when it is derived from an earlier version. Higher levels of abstraction may be needed to identify the similarities.

At those higher levels of abstraction, copyright protection:

- DOES NOT extend to any ideas, procedures, processes, systems, methods of operation, concepts, principles, or discoveries contained in the original program.

- MAY extend beyond the literal code of a program to its nonliteral aspects, such as its architecture, structure, sequence, organization, operational modules, and computer user interface.

These more abstract similarities are not always obvious to the naked eye; identifying them often requires expert guidance. In any event, once the noncopyrightable similarities are filtered out, only the remaining copyrightable similarities are compared. Substantial similarity of the copyrightable elements is evidence of copyright infringement, but substantial similarity of the noncopyrightable elements means nothing at all.

In Chapter 6, in the context of the GPL, I described the arguments that have raged in the open source community about whether linking between programs creates a derivative work. Nothing in the law of copyright suggests that linking between programs is a determinative factor in derivative work analyses by courts—except perhaps as evidence of one of the abstract, nonliteral, copyrightable aspects of the software, such as program architecture, structure, and organization.

In such cases, the burden usually rests on the licensor to explain to the court why the simple interaction of software modules—black boxes merely plugged into other software—creates a derivative work of the black boxes. Merely combining black boxes, I suggested earlier, creates collective works, not derivative works.

Substantial similarities, standing alone, are never enough to characterize a derivative work. An independent creation is not a derivative work no matter how much it resembles a preexisting work. Copyright only protects against copying, not against someone writing the same expression independently, by coincidence. So plaintiffs may still have to prove actual copying.

Evidence can sometimes be provided by a plaintiff to show that an alleged infringer had access to the preexisting work and an opportunity to copy it. For open source software, proving access and opportunity is relatively easy because the source code is published. The burden of proof then may shift to the defendant to show that the substantial similarities were an accidental byproduct of independent creation.

In practice, most infringing derivative works of software are blatant and not subtle because it usually takes more time to obscure an infringing work than to just write it anew from scratch. Nevertheless, when defendants intentionally set out to hide their copyright infringement, it can be difficult to prove.

Such extreme efforts to cheat open source software licensors by pretending not to have created derivative works is usually a waste of time. It is often less expensive just to write equivalent software from scratch. Why risk creating software with questionable provenance? It may result in an expensive infringement lawsuit—which you may lose. If you try to sell such software, your customers may reject it as risky even though it is not actually proven to have infringed.

The better plan is not to tread too close to the line separating collective and derivative works. Companies that create software should make sure their employees don't have access to preexisting software, and they should train their employees not to copy other software.

Instead of accepting the risk that their software will be called a derivative work, companies sometimes prefer to refuse software under licenses containing reciprocity obligations. Such software may then be available under dual licensing options, such as the ones described in Chapter 11.

Instead of avoiding the creation of derivative works, there's a more principled argument to be made that it is a public benefit to create derivative works and to distribute them under reciprocal open source licenses. That way everyone can profit from improvements to software.

Contributions to the software commons are always welcomed. So I encourage you to take a very broad view of your reciprocity obligations; don't be stingy about them. Contribute as many of your derivative works as possible to the community.

Patent Infringement Litigation

Patent infringement claims usually appear unexpectedly. They are serious matters, expensive, and potentially very damaging. When faced with a claim of infringement, you should consult an attorney. Fighting patent infringement litigation on your own is foolish.

You can't prevent patent infringement lawsuits, but your licenses can help you defend against them. Some open source licenses have very strong patent defense provisions (e.g., GPL section 7, MPL sections 8.2 and 8.3, CPL section 7, OSL/AFL section 10). These defensive termination provisions act

by increasing the cost of suing an open source licensor for patent infringement. If the licensed software has value to the patent owner, he or she may prefer to forgo a patent infringement lawsuit rather than lose the license to the software.

Defensive termination provisions help protect open source licensors from infringement lawsuits by their own licensees. But there is no possible license provision that can protect a licensor—or anyone else—from lawsuits by third parties who are not licensees.

A collective approach to patents can also be helpful to encourage open source and proprietary software development. That is why companies cooperate, within the limitations of the antitrust law, to develop industry standards that are unencumbered by patents. The important role of open standards for the success of open source is the topic of the next and final chapter of this book.

SCO *vs.* Open Source

Anyone who has read the earlier section on standing will quickly recognize the incongruity of the title "SCO *vs.* Open Source." SCO is shorthand for The SCO Group, Inc., a Delaware corporation. *Open source* is a software development, business, and licensing model. Open source does not have standing to be a defendant in a lawsuit. Neither SCO, nor any other plaintiff, can sue an entire movement—particularly one that is so thoroughly grounded in intellectual property and contract law—over any cause of action worth litigating.

As this is written, The SCO Group is a party to several rancorous lawsuits against certain specific software companies, including IBM, Novell, and Red Hat, over intellectual property rights in the flagship open source product, Linux.

Initially, SCO's complaint alleged that it had licensed certain proprietary software to IBM and that IBM had then improperly contributed that software to open source Linux. The original lawsuit was framed in traditional breach of contract terms as a dispute over an agreement between IBM and SCO that purportedly required IBM to maintain the trade secret status of certain software licensed to it by SCO. IBM denied all material allegations and then, in turn, cross-complained against SCO, alleging breach of contract and patent infringement. SCO has since broadened its complaint to include allegations about the GPL under which Linux is licensed.

Then Red Hat sued SCO, alleging unfair business practices, among other business torts. Finally, SCO and Novell disputed the terms of the original contract under which SCO's predecessor-in-interest bought certain rights to UNIX from Novell.

The SCO litigation may be resolved by the time you read this book, in which event use the following opinion as a way of evaluating my prescience: The SCO cases are a legal mess, an unfortunate opportunity for companies to spend millions of dollars in attorneys' fees to defend their intellectual property and contractual rights and to argue about enormous damage claims. But they don't directly affect open source. All the licenses described in this book and all the software licensed under those licenses—with the possible exception of some small portion of Linux—will remain valid no matter what happens in the SCO lawsuits. As to that small portion of Linux, it may turn out after litigation to be no portion of Linux at all.

Like any other person, SCO has rights only to copyrightable works that it authored or acquired by assignment or license. The independently created copyrightable works of others, such as the contributions to Linux by Linus Torvalds

and thousands of other programmers worldwide, are not owned by SCO. Nor can SCO own the unpatented ideas embodied in Linux. Given what I know about the history and evolution of operating systems (including UNIX and Linux), it is inconceivable to me that significant portions of Linux are copies or derivative works of any SCO software. Most Linux experts reassure me that, when the dust of this litigation settles, the courts will determine that SCO owns little or nothing of the intellectual property in Linux.

The SCO lawsuit reveals some interesting open source ironies. SCO itself distributed Linux open source software and, even after SCO had filed its first complaint against IBM, licensees could still obtain Linux under the GPL from an SCO website. I'm not aware of any important case—and Linux software is *important* in this sense—where commercially sophisticated licensors have been allowed to disavow their own licenses for the very software under dispute.

SCO's public arguments challenging the constitutionality of the GPL are particularly intriguing. (See the Open Letter from Darl McBride, president and CEO of SCO, dated December 4, 2003.) It would be truly exciting news if U.S. courts allowed a company to challenge the constitutionality of its own license.

But suppose the courts finally do step back from this entire open source phenomenon and ask, in the context of a legitimate lawsuit by parties with standing: "Is this licensing scheme to build a commons of open source software constitutional? Should licensors be allowed to turn copyright on its head this way, conditioning a license to software on a reciprocal obligation to contribute?"

There is absolutely no legal basis to argue that this scheme is unconstitutional. It is a basic legal principle that licensors can

do what they wish with their intellectual property and set conditions for its use.

The public excitement about the SCO cases proves the point I've hinted at throughout this chapter. Litigation about open source software will be rare; if it were a common occurrence the public would be bored with the rather hysterical SCO litigation claims by now. The uniqueness of the SCO litigation, and its multi-billion dollar damage claims, makes it stand out.

The SCO litigation against Linux also marks a maturation of the open source movement, which is finally a big enough phenomenon for its software to be the object of a big lawsuit. Put simply, open source software is now important enough to sue over. The popularity and success of open source software and of Linux in particular inevitably draw litigation because there are important and valuable economic interests at stake.

The SCO litigation is an aberration. It is a big lawsuit about what most knowledgeable attorneys believe is a small issue between particular companies. It will eventually be resolved—and Linux and open source will continue to evolve. This too shall pass.

Many open source advocates have secretly longed for test cases so that the courts can clearly articulate the laws of open source licenses. There have thus far been very few such cases. Open source parties argue mostly about breach of contract, trademark infringement, occasionally patent infringement, and whether a derivative work has been created. Most such arguments are resolved informally, as is true for almost all commercial disputes in most civilized countries. Why would open source licensors and licensees sue each other if they can work out differences in a spirit of open source generosity?

It is difficult to imagine an important case where open source licensors and licensees will litigate about free software.

As long as open source projects act as responsible custodians of intellectual property, keeping careful track of the software they receive and the software they create, then licensees can rely on the continued availability of that software under open source rules. And as long as licensees honor the conditions of the licenses for software they accept, there is little reason to fear it will be taken away through litigation.