SOFTWARE COMPANIES HAVE mass-marketed computer programs for the past few decades on terms that typically purport to restrict the right of end users to resell or otherwise transfer their interests in copies of software they have purchased. The restrictions are usually stated in documents known as shrink-wrap or click-through “licenses.” Vendors of other types of digital content sometimes distribute their works with similar restrictions.

Shrink-wraps are documents inserted in packaged software, often just under the clear plastic wrap surrounding the package, informing purchasers they are not owners of copies of programs they just bought, but instead have rights in the program that are limited by the terms of a license agreement. Click-throughs are similar in substance, although the “license” terms only become manifest when you try to install the software and are directed to click “I agree” to certain terms.

Most of us ignore these documents and the restrictions they contain. Because we bought our copy of that software, we think we own it, regardless of what any “license” document says. We are also quite confident the vendor won’t take any action against us, even if we do violate one of the terms, because realistically the vendor can’t monitor every end user of its products.

The debate over whether mass-market transactions like these are really “sales” of goods, notwithstanding the “license” label, has been going on for decades. Strangely enough, it has yet to be definitively resolved. A recent appellate court ruling has upheld the license characterization, but a further appeal is under way in that case. This ruling is also at odds with other appellate court decisions. So things are still up in the air on the ultimate issue. This column will explain what is at stake in these battles over your rights in your copies of purchased software.

Copyright law allows rights holders to control only the first sale of a copy of a protected work to members of the public.

Three Legal Options

The distinction between “sales” and “licenses” really matters when assessing the risk of liability to copyright owners if resale restrictions are ignored.

Copyright law allows rights holders to control only the first sale of a copy of a protected work to members of the public. After the first sale, the owner of that copy is entitled to resell or otherwise transfer (for example, give it away as a gift or lend it to others) the copy free from risk of copyright liability. Bookstores and libraries are among the institutions made possible by copyright law’s first-sale doctrine.
What happens if copyright owners try to restrict resales through license restrictions? There are three possible outcomes.

First, the effort to restrict resales may be deemed a nullity, as it was in a famous 1908 Supreme Court case, *Bobbs-Merrill Co. v. Straus*. Bobbs-Merrill sold books to Straus containing a prominent notice that resale of the books except at a stated price would be treated as copyright infringement. When Straus sold the books for a lower price, Bobbs-Merrill sued for infringement. The Court refused to enforce this resale restriction, saying that the copyright owner was entitled to control only the first sale of copies of its works to the public.

Second, a resale restriction may be enforceable against the purchaser insofar as he has agreed not to resell his copy, but it would be unenforceable against anyone to whom the purchaser might subsequently sell his copy.

This result might seem odd, but there is a fundamental difference between contract and property rights: Contracts only bind those who have agreed to whatever terms the contract provides; property rights create obligations that are good against the world.

A first-sale purchaser may thus have breached a contractual obligation to the copyright owner if it resells its copy of the work in violation of a resale restriction, but he is not a copyright infringer.

Those who purchase copies of copyrighted works from owners of first-sale copies are not at risk of either copyright or contract liability. These third-party purchasers are also free to resell their copies to a fourth party without fear that either is at risk of any liability to the copyright owner.

Third, courts may rule that the first-sale rule does not apply to mass market “license” transactions involving copies of copyrighted works because no “sale” has taken place. Under this interpretation, secondary markets in those copies are illegal. Anyone who purports to resell the copies is a copyright infringer for distributing copies of copyrighted works without getting permission from the copyright owner.

**Vernor and Augusto**

My March 2009 Legally Speaking column (“When is a ‘License’ Really a Sale?”) discussed two lower-court decisions in which copyright owners challenged the resale of copies of copyrighted works on eBay. In both cases, the courts ruled that copyright’s first-sale rule applied, notwithstanding transfer restrictions, because of economic realities of the transactions.

The plaintiff in *Vernor v. Autodesk* asked the court to declare that he was the owner of copies of Autodesk software he purchased from one of Autodesk’s customers and that he was entitled under the first-sale doctrine to resell those copies on eBay.

Autodesk claimed no sale had taken place because the software was licensed on terms that forbade transfer of the copy to third parties. Autodesk asked the court to declare that sales of these copies on eBay constituted copyright infringement.

*UMG v. Augusto* involved promotional CDs of music. Augusto purchased these CDs at flea markets, online auctions, and used CD stores. Language on the CD packaging indicated they were licensed for personal use only and could not lawfully be sold or otherwise transferred to third parties. When Augusto started selling UMG promotional CDs on eBay, UMG sued him for infringement.

My March 2009 column predicted that Autodesk and UMG would appeal the trial court rulings against them, and that the software industry could be expected to push very hard for a reversal, particularly in the *Vernor* case.

The Ninth Circuit Court of Appeals heard the arguments in *Vernor* and *Augusto* on the same day. In September 2010, the appellate court ruled in favor of Autodesk. Yet it upheld Augusto’s first-sale claim.

In assessing whether the first-sale rule should apply to mass-market transactions like these, it is useful to compare the economic realities test used by the trial courts in the *Vernor* and *Augusto* cases and the labeling and restrictions test adopted by the Ninth Circuit in *Vernor*.

**Economic Realities Test**

Under this test, a copyright owner’s characterization of a transaction as a license, rather than a sale, is not dispositive. It is instead only one factor among many that should be weighed in determining the true nature of the transaction.

Other factors include whether the purchaser has the right of perpetual possession of the copy, whether the rights holder has the right and ability to reclaim the copy if the license terms are violated, whether the purchaser has paid substantial sums for the privilege of permanent possession, and whether the purchaser has the right to discard or destroy the copy. The marketing channels through which the copy was obtained (such as purchasing packages of software at Walmart or Office Depot) may also be relevant.

Under the economic realities test,
Vernor and Augusto seem to be owners of copies. Those from whom they obtained the products had, it seems, the right of perpetual possession in the copies, and they could destroy or discard the copies if they wished. The software in Vernor had been purchased through a mass-market transaction, and the CDs in Augusto had been mailed for free to people who had not requested the CDs and indeed, UMG had not even kept track of the persons to whom the promotional CDs had been sent.

In a previous case, U.S. v. Wise, the Ninth Circuit reversed a conviction for criminal copyright infringement because the actress from whom Wise obtained a copy of a movie was the owner of that copy, notwithstanding various restrictions on what she could do with the copy, including transfers to third parties.

In Vernor’s petition for rehearing by the full Ninth Circuit Court of Appeals, he argues the Ninth Circuit’s ruling is in conflict with Wise and with precedents from other appellate courts, including Bobbs-Merrill.

Labeling and Restrictions Test
The Ninth Circuit in Vernor relied in part on MAI Systems Corp. v. Peak Computer, in which a Ninth Circuit panel in 1993 ruled that customers of Peak’s computers, on which Peak software was installed, were not owners of copies of this software, but rather licensees. Owners of copies of copyrighted software are entitled to make copies for their use and to authorize third parties to make use-copies; non-owners are not entitled to this privilege.

MAI provided maintenance services for Peak computers to Peak customers. When MAI technicians turned on Peak computers to service them, they made temporary copies of Peak software in the random access memory. Peak argued, and the Ninth Circuit agreed, that these copies were infringing because they were not authorized by Peak.

MAI v. Peak cited no authority and offered no analysis in support of its ruling that Peak’s customers were non-owners of their copies of Peak software. Peak’s characterization of the transaction as a license was, for that panel, dispositive.

The three-judge panel decision in Vernor did not rely on the license label alone as a basis for rejecting Vernor’s first-sale argument. But the license label was, as in MAI v. Peak, given considerable weight. The court directed that two other factors be taken into account: whether the license restricted transfers of the copies and whether it contained other substantial restrictions. The panel ruled that Autodesk should prevail against Vernor under this test. The restrictions in Augusto, by contrast, were less substantial than those in Vernor.

Conclusion
Software companies have been cheered by the Ninth Circuit’s ruling in Vernor. But the rest of us should be worried about its implications. Think about what Vernor may mean for flea markets, bookstores, libraries, garage sales, and auction sites. Even selling a used computer loaded with software is infringing on this theory. Think also about how easy it is for a vendor to put a “license” label on a mass-marketed product with copyrighted or patented components that states that any transfer of that copy to third parties will subject the transferor to copyright or patent infringement charges.

Consumers enjoy significant benefits from the existence of secondary markets. The first-sale limit on patent and copyrights is essential to the operation of those markets. Vernor and Augusto’s cases are important to the future of competition in product markets and to preservation of the long-standing balancing principle in copyright law that the first-sale rule represents.

Pamela Samuelson (pam@law.berkeley.edu) is the Richard M. Sherman Distinguished Professor of Law and Information at the University of California, Berkeley. Copyright held by author.
Software’s close encounters with the law provide some lessons for our future.

BY JAMES BOYLE

What Intellectual Property Law Should Learn from Software

TWENTY-FIVE YEARS AGO a vigorous debate raged in U.S. legal academia over whether software should be covered by patent or copyright or some third option. (Pamela Samuelson, who writes regularly in Communications, was co-author of the best article on the subject.6) In practice, software ended up being covered by both schemes, partly due to actions by the U.S. Congress, including several references to software in the Copyright Act, and partly as a result of decisions by the Copyright Office, the Patent and Trademark Office (PTO), and by judges. One could copyright one’s code and also gain a patent over the “non-obvious” novel and useful innovations inside the software. (In much of the rest of the world, software also came to be covered by copyright, though the status of patents over software was sometimes more obscure.)

What can we learn from the history of the years since? A lot, it turns out, some not limited to the U.S., where intellectual property law often tends (for better or for worse) to disproportionately influence technology policy worldwide.

At first, the use of copyright stirred the most concern. Copyright is built around an assumption of diverging innovation, the fountain or explosion of expressive activity. Different people in different situations who sit down to write a sonnet or love story, it is presumed, will produce very different results rather than be drawn to a single result. Thus, strong rights over the resulting work are not supposed to inhibit future progress. I can find my own muse, my own path to immortality. Creative expression is presumed to be largely independent of the work...
of prior authors. Raw material is not needed.

There are lots of reasons to doubt that this vision of “creation out of nothing” works very well, even in the arts, the traditional domain of copyright law. But whatever its merits or defects in the arts, it seems completely wrong-headed when it comes to software. Software solutions to practical problems do converge, and programmers definitely draw upon prior lines of code. Worse still, software tends to exhibit “network effects.” Unlike my choice of novel, my choice of word-processing program is strongly influenced, perhaps dominated, by the question of what program other people choose to buy. That means that even if a programmer could find a completely different way to write a word-processing program, this programmer has to be able to make it read the dominant program’s files and mimic its features if the programmer is to attract any customers at all. This hardly sounds like completely divergent creation.

Seeing the way software failed to fit this Procrustean bed of copyright, many scholars presumed the process of forcing it into place would be catastrophic. They believed that, lacking patent’s high standards, copyright’s monopolies would proliferate. Copyright’s treatment of follow-on, or “derivative,” works would impede innovation, it was thought. The force of network effects would allow the copyright holder of whatever software became “the standard” to extract huge monopoly rents and prevent competing innovation for many years longer than the patent term. Users of programs would be locked in, unable to shift their documents, data, or skills to a competing program. Doom and gloom abounded among copyright scholars, including many who shared the premise that software should be covered by property rights. They simply believed that these were the wrong property rights to use.

Copyright did indeed cause problems for software developers, though it is difficult to judge whether they outweighed the economic benefits of encouraging software innovation, production, and distribution. But the negative effects of copyright were minimized by a remarkably prescient set of actions by courts and, to a much lesser extent Congress, so the worst scenarios did not come to pass. Courts interpreted the copyright over software narrowly, so it covered little beyond literal infringement. They developed a complicated test to work out whether one program infringes the details of another program. The details give law students headaches, but the effects were simple. If your software is similar to mine merely because it performed the same function or because I picked the most efficient way to perform some task or even because there was market demand for doing it that way, then none of those similarities counted for the purposes of infringement. Nor did material that was taken from the public domain. The result was that while someone who made literal copies of Windows Vista was clearly infringing copyright, the person who made a competing program generally would not be.

In addition, courts interpreted copyright’s fair-use doctrine to cover something called “decompilation,” basically taking apart someone else’s program so you can understand and compete with it. As part of the process, the decompiler had to make a copy of the program. If the law were read literally, decompilation would hardly seem a fair use. The decompiler makes a whole copy, for a commercial purpose, of a copyrighted work, precisely in order to cause harm to its market by offering a substitute good. But the courts took a broader view. The copy was a necessary part of the process of producing a competing product, rather than a piratical attempt to sell a copy of the same product. This limitation on copyright provided by fair use was needed in order to foster the innovation that copyright is supposed to encourage.

These rulings and others like them meant that software was protected by copyright but also that the copyright did not give its owner the right to prevent functional imitation and competition. Is that enough? Clearly the network effects are real. Most of us use Windows and Microsoft Word, and one very big reason is because everyone else

---

b See, for example, Sega Enters. Ltd. v. Accolade Inc., 977 F.2d 1510 (9th Cir. 1992).
does. Optimists believe that the lure of capturing this huge market will keep potential competitors hungry and monopolists scared. The lumbering dominant players, goes the argument, will not become complacent about innovation or try to grab every morsel of monopoly rent. They still have to fear their raptor-like competitors lurking in the shadows. Perhaps. Or perhaps it also takes the consistent threat of antitrust enforcement. In any event, whether or not we hit the optimal point in protecting software with intellectual property rights, these rights certainly did not destroy the industry. It appeared that, even with convergent creativity and network effects, software could be crammed into the Procrustean bed of copyright without killing it off in the process. Indeed, to some, it seemed to fare quite well. They would claim that the easy legal protection provided by copyright gave a nascent industry just enough protection to encourage the investment of time, talent, and dollars, while not prohibiting the next generation of companies from building on the innovations of the past.

In addition, the interaction between copyright and software has produced some surprising results. There is a strong argument that it is the fact that software is copyrightable that has enabled the “commons-based creativity” of free and open source software. What is commons-based creativity? Basically it is creativity that builds on an open resource available to all. An additional component of some definitions is that the results of the creativity must be fed back into the commons for all to use. Think of English. You use it without license or fee, and you innovate by producing new words, slang, or phrases without clearance from some Académie Anglaise. After you coin your term, it is in turn available to me to build upon or use in my own sentences, novels, or jokes. And so the cycle continues. But with words we have commons-based creativity because there were no property rights over the relevant material. The software commons is different.

The creators of free and open source software were able to use the fact that software is copyrighted and that the right attaches automatically on creation and fixation to set up new distributed methods of innovation. For example, free and open source software under the General Public License (such as Linux) is a “commons” to which all are granted access. Anyone may use the software without restriction. All are guaranteed access to the human-readable source code, rather than just the inscrutable machine code, so they can understand, tinker, and modify. Modifications can be distributed so long as the new creation is licensed under the open terms of the original. This creates a virtuous cycle whereby each addition builds on the commons and is returned to it. The copyright over the software is the “hook” that allowed software engineers to create a license that gave free access and the right to modify, and required future programmers to keep offering these freedoms. Without the copyright, those features of the license would not have been enforceable. For example, someone could have modified the open program, releasing it without the source code, thus denying future users the right to understand and modify easily. To use an analogy beloved of free-software enthusiasts, the hood of the car would be welded shut. Home repair, tinkering, customization, and redesign become practically impossible.

If there were no copyright over software at all, software engineers would have other freedoms, even if not legally guaranteed open access to source code. Still, it is difficult to deny that the extension of the property regime had—bizarrely, at first sight—actually enabled the creation of a continuing open commons. The tempting real-estate analogy would be environmentalists using strong property rights over land to guarantee conservation and open access to a green space, whereas without property rights, the space could be despoiled by all.

So much for copyright. What about patents? U.S. patent law had customarily drawn a firm line between patentable invention and unpatentable idea, formula, or algorithm. The mousetrap could be patented, but not the formula used to calculate the speed at which it snaps shut. Ideas, algorithms, and formulae were in the public domain, as were “business methods.” Or so we thought.

The line between idea or algorithm on the one hand and patentable machine on the other looks nice and easy. But put that algorithm into a computer and things begin to look more complex. Say, for example, the algorithm was the process for converting miles into kilometers and vice versa. In the abstract, this is classic public-domain
stuff, no more patentable than $E = mc^2$ or $F = ma$. What about when those steps are put onto the tape of the Turing machine, onto a program running on the hard drive of a computer?

For some time, the U.S. Court of Appeals for the Federal Circuit (the leading patent court in the U.S.) has seemed to believe that computers can turn an unpatentable idea into a patentable machine. In fact, in this conception, the computer sitting on your desk becomes multiple patentable machines—a word-processing machine, an email machine, a machine running the program to calculate the tensile strength of steel. I want to stress that the other bars to patentability remain. My example of miles-to-kilometers conversion would be patentable subject matter, but, we hope, no patent would be granted because the algorithm is not novel and is obvious. (Though sadly, the PTO seems determined to undermine this hope by granting patents on the most mundane and obvious applications; two excellent books by Besson and Meurer and by Jaffe and Lerner explore this point, as well as other deeper problems with the patent system.) But the concern here is not limited to the idea that, without a subject-matter bar, too many obvious patents will be granted by an overworked and poorly incentivized patent office. It is that the patent was supposed to be granted at the very end of a process of investigation and scientific and engineering innovation. The formulae, algorithms, and scientific discoveries on which the patented invention was based remained in the public domain for all to use. It was only when we got to the very end of the process, with a concrete innovation ready to go to market, that the patent was to be given. Yet the ability to couple the abstract algorithm with the concept of a Turing machine undermines this conception. Suddenly the patents are available at the very beginning of the process, even to people who are merely specifying, in the abstract, the idea of a computer running a particular series of algorithmic activities.

The words “by means of a computer” seem to be an incantation of magical power, able to transubstantiate the ideas and formulae of the public domain into private property. And, like the breaking of a minor taboo that presages a Victorian literary character's slide into debauchery, once that first wall protecting the public domain was breached, the courts found it easier and easier to breach still others. If one could turn an algorithm into a patentable machine (by simply adding “by means of a computer”), then could one not turn a business method into something patentable by specifying the organizational or information technology structure through which the business method is to be implemented?

You might wonder why we would want to patent business methods. Intellectual property rights are supposed to be handed out only when necessary to produce incentives to supply some public good, incentives that otherwise would be lacking. Yet there are already plenty of incentives to come up with new business methods. (Greed and fear are the most obvious.) There is no evidence to believe we need a state-backed monopoly to encourage the development of new business methods. In fact, we want people to copy the businesses of others, lowering the price as a result. The process of copying business methods is called “competition” and is the basis of a free-market economy. Yet patent law would prohibit it for 20 years. So why introduce patents? Brushing aside such minor objections with ease in 1998, in a case called State Street, the Court of Appeals for the Federal Circuit declared business methods to be patentable. Could this really be what Thomas Jefferson had in mind when he said “I know well the difficulty of drawing a line between the things which are worth to the public the embarrassment of an exclusive patent, and those which are not”? I doubt it.

In 2008, the Court of Appeals for the Federal Circuit revisited this ruling in a case called In re Bilski. Perhaps made wary by several spankings they had recently received at the hands of the U.S. Court, the Court of Appeals for the Federal Circuit declared business methods to be patentable.
look at the purpose of the law they are enforcing when seeking to understand what it means. In areas of regulation that are obviously “instrumental”—aimed at producing some particular result in the world—this approach is ubiquitous. In applying the antitrust laws, for example, courts have given meaning to the relatively vague words of the law by turning to economic analysis of the likely effects of different rules on different market structures.

Patent law is as instrumental a structure as one could imagine. In the U.S., for example, the constitutional authorization to Congress to make patent and copyright legislation is very explicit that these rights are to be made with a purpose in view. Congress has the power “to promote the progress of science and useful arts, by securing for limited times to authors and inventors the exclusive right to their respective writings and discoveries.”j One might imagine that courts would try to interpret the patent and copyright laws with that purpose firmly in mind. Yet utilitarian caution about extending monopolies is seldom found in the reasoning of the U.S.’s chief patent court. Until Bilski, the court had preferred to quote a phrase from a congressional report that patentable subject matter includes “anything under the sun that is made by man.”k

The difference is striking. Jefferson said that the job of those who administered the patent system was to see if a patent was worth the embarrassment to the public before granting it. The Constitution tells Congress to make only those patent laws that “promote the progress of science and useful arts.” One might imagine that this constitutional goal would guide courts in construing the same laws. Yet in our chief patent court for the past 20 years, neither Jeffersonian ideals nor the constitutional text has seemed relevant to its thinking when interpreting statutory subject matter. Anything under the sun that is made by man is patentable subject matter, and there’s an end to it. The case that announced the rule on business methods involved a patent over

The words “by means of a computer” seem to be an incantation of magical power, able to transubstantiate the ideas and formulae of the public domain into private property.

---

g State Street, 149 F.3d at 1373.
h In re Bilski, 545 F.3d at 998 (Newman, J., dissenting).
i In re Bilski, 545 F.3d at 998, 1007 (Mayer, J., dissenting).
the process of keeping accounts in a “hub-and-spoke” mutual fund, including multiplying all of the stock holdings of each fund in a family of funds by the respective current share price to get total fund value, then dividing by the number of mutual-fund shares that each customer actually holds to find the balance in their accounts. As my son observed, “I couldn’t do that until nearly the end of third grade!”

In theory of course, if the patent is not novel or obvious, it will still be refused. The Supreme Court recently held that the Court of Appeals for the Federal Circuit made “non-obvious” too easy a standard to meet. It is unclear, however, whether this judgment will produce concrete effects on actual practices of patent grants and litigation. The PTO’s system puts pressure on examiners to issue patents, and it is very expensive to challenge those that are granted. Better would be, where possible, to rule out certain subject matter (such as business methods) in the first place and more narrowly craft software patents so as to avoid the dangers the copyright decisions anticipated so clearly. Judge Mayer is right. Tempted in part by the power of the metaphor of “idea made machine” in the context of a computer, the Court of Appeals for the Federal Circuit has not been able to bring itself to do so. Where copyright law evolved to walk off, encyst, and minimize the dangers of extending protection over software, patent law initially extended the idea behind software patents to make patentable any thought process that might produce a useful result. Even when it got rid of the “useful result” language, the court was unable to bring itself to declare business methods unpatentable. Once breached, the walls protecting the public domain in patent law show a disturbing tendency to erode at an increasing rate.

To sum up, the conceptual possibilities presented to copyright and patent law by software were fascinating. Should we extend copyright or patent to cover the new technology? The answer was “We will extend both!” Yet the results of the extension were complex and unexpected in ways we should try to understand if we want to predict the effect of intellectual property on future technologies. Who would have predicted that software copyrights could be used to create a self-perpetuating commons, as well as a monopoly over operating systems, or that judges would talk knowingly of network effects in curtailing the scope of coverage? Who would have predicted that patents would be extended not only to basic algorithms implemented by a computer but to methods of business themselves? (Truly, a strange return to legalized business monopolies for a country whose founders viewed them as one of the greatest evils that could be borne.) The rest of the world has (wisely) been resistant to granting patents over business methods, and even to so-called “pure” software patents. (The empirical evidence, of which there is far too little, suggests that expansive software patents may actually have a negative effect on research and development.) Yet as global legal harmonization sweeps onward, little attention is being paid to empirical evidence, and it is not clear which way the norms will tip. Our attitude should be to demand rigorous empirical and economic study before we create or extend legal monopolies. Expansive new rights over emerging technologies may be necessary to encourage innovation, but the case must be made on facts, not faith.

What can we learn from this history? First, we should realize that the mere decision to include a technology within a property regime is only the first in a sequence. As the copyright system showed with software, it is possible to trim protection so as to minimize overreaching. As the business-method patent decisions show us, we don’t always do it. Second, we should understand that we have some new methods of combining property rights and an open “commons” of raw material. The experience of free and open source software should be studied to see whether it has implications for new technologies. We need all the innovation tools we can get. Third, we should be mindful of the fact that much depends on the moment in the development of a technology when property rights begin to be rigorously applied. For better or for worse, property rights came fully to software at a point when no one would have thought of claiming the most fundamental building blocks—patenting the idea of a Turing machine or the precepts of Boolean algebra. The basics of the field were there for all to build upon. Will that be true with future technologies?

It is disquieting to realize that today the answer to this question is very difficult to provide. In one particular area, synthetic biology, which shares aspects of both software (programming in genetic code) and genetic engineering, there is considerable reason for alarm. As my colleague Arri Rai, and I note in an article on the subject, it is quite possible to imagine a perfect storm in which the expansive patent law decisions of the past 20 years do to synthetic biology what they could not do to software—lock up the basic building blocks before the field can develop.

The fundamental ideas behind our intellectual property system are sound. Intellectual property rights can be important, even vital, for the development of a particular area of technology. But it is just as easy to harm innovation with rights that are too strong as too weak. The example of software could teach us a lot about the future of good intellectual property policy in high technology, but first we need to pay attention to it.

References

James Boyle (boyle@law.duke.edu) is William Neal Reynolds Professor of law at Duke Law School, Durham, NC. This article is adapted from his book *The Public Domain: Enclosing the Commons of the Mind* (2008 by Yale University Press), which is freely downloadable from http://thepublicdomain.org under a Creative Commons license.

Copyright held by author.
Damage Control

The U.S. patent system is overdue for reform, but what needs fixing, and how, is a matter of some dispute.

In 2006, faced with the threat of a court-ordered shutdown, Research In Motion (RIM), the Canadian manufacturer of BlackBerry phones, reached a settlement on a prolonged and vicious patent dispute with Virginia-based NTP. Although NTP’s threat of injunction was widely viewed as extortion in the IT industry (the company neither makes nor sells any products, and its primary assets are the patents of the late inventor Thomas Campana), RIM nonetheless agreed to pay $612.5 million in a “full and final settlement of all claims.”

That figure sparked both disbelief and outrage among many members of the IT industry. It also increased calls for substantial legislative reform. Patent law in the U.S. has not been substantially updated since 1952, and is frequently thought to be out of sync with modern business practices. However, exactly what needs reforming, and how, is a matter of some dispute.

Most companies and entrepreneurs agree with the principles that underlie the U.S. patent system, which fosters innovation by granting inventors an exclusive, though temporary, right to their creations in exchange for sharing their work. But what constitutes a patentable invention, and how should it be protected? Critics complain that American patents are too easy to file—the U.S. Patent and Trademark Office (USPTO) grants tens of thousands of patents each year—and too easy to defend in expensive legal suits. (The NTP patents that RIM was found to have infringed upon might never have been granted in many countries.) Companies whose livelihoods depend on revenue from patent licenses, on the other hand, are loath to support anything that might weaken the value of their portfolio. The Patent Reform Act of 2007, a reform bill introduced by Senators Patrick Leahy (D-VT) and Orrin Hatch (R-UT) and Representatives Howard Berman (D-CA) and Lamar Smith (R-TX), stalled last April as legislators were unable to reconcile these competing interests.

Trolling for Dollars

It is impossible to make a piece of electronic technology without relying on dozens, if not thousands, of individually patented components. In 2003, one computer hardware firm told the U.S. Federal Trade Commis-
sion that more than 90,000 patents, held by some 10,000 parties, were related to a single microprocessor. Most large IT companies, realizing that their products may regularly infringe on the patents of their rivals, and vice versa, have struck an implicit truce to keep themselves out of court. Increasingly, however, they have come under fire from so-called “non-practicing entities” (known in the IT industry by the less-charitable moniker “patent trolls”), companies whose primary line of business is the litigation of patent lawsuits.

Trolls purchase patents from inventors and other sources—such as bankrupt companies that are selling off their assets—then sue for infringement, hoping to cash in on settlements and royalties. It’s not clear how many patent trolls currently exist, but there’s no doubt they’ve had an impact on the IT industry. In 2005, Yahoo! was engaged in four patent-related lawsuits; by 2008, the number had swollen to 22. All are plaintiff-driven cases, and most of them were filed in the Eastern District of Texas (a jurisdiction widely considered friendly to patent holders), in spite of the fact that Yahoo! has neither offices nor servers nearby. According to Joseph Siino, Yahoo!’s vice president of intellectual property, each lawsuit costs the company an estimated $2 to $5 million to defend. Other IT firms report similar figures. Terry Alberstein, director of corporate affairs for Cisco, says that each of the 30 active patent-related suits the company currently faces was brought by a non-practicing entity. The cost of litigation: a staggering $30 to $50 million a year.

How to fight this dramatic drain on resources? A cross-industry organization called the Coalition for Patent Fairness, with members such as Dell, Hewlett-Packard, and Intel, was created in 2006 to lobby for litigation reform; a handful of other organizations have taken up the cause, as well. Among the most interesting is the newly formed Allied Security Trust, which seeks to fight patent trolls by acquiring patents and granting member companies nonexclusive rights to use them.

One thing reformers hope to see is a tightening of the standards used to determine patentability. The USPTO is overworked and understaffed, and examiners frequently grant patents to inventions that do not actually merit them, according to some critics. But the real battles happen in court, where a company’s best defense against a claim of infringement is usually to argue that the patent under question is invalid. To be patentable, an invention must be novel, useful, and “non-obvious” to an expert in the field. It is this last requirement that has proven troublesome: what is obvious? Until recently, the courts held an invention to be obvious only if it was explicitly figured by prior “teaching, suggestion, or motivation.” On the other hand, as critics point out, when something is obvious, few people write it down.

The stalled Patent Reform Act tried to address the problem through a controversial post-grant review process that would have made it easier to challenge the validity of a patent. Currently, patents can be challenged through special re-examination proceedings at the USPTO, which are widely seen as cumbersome and ineffectual, or by litigation. The Patent Reform Act would have created a three-judge tribunal whose sole purpose would be to consider patent validity in a less costly and more efficient fashion than in court. Though many companies supported the tribunal’s creation, others worried it would subject patents to potentially endless challenges unless temporal limits were applied to its jurisdiction. Just how many months or years after the granting of a patent the tribunal should be able to rule on a patent’s validity was the subject of fierce arguments.

Despite these legislative battles, the courts have begun to reconsider the issue of obviousness on their own. The U.S. Supreme Court reset the bar last spring when it ruled that the combination of two existing technologies was not “non-obvious” and thus did not deserve a patent. Welcomed by many in the IT industry as an important step toward reform, the decision also granted examiners and courts more discretion to use “common sense” when determining the obviousness of...

---

**Virtual Reality**

### Creating ‘Virtual’ Objects With Ultrasound

A team of Japanese researchers has demonstrated a system that uses ultrasound waves to create “virtual” objects in mid-air, BBC News reports. The system, developed by Takayuki Iwamoto and colleagues from the University of Tokyo, uses ultrasonic transducers, which produce ultrasound. As sound is a pressure wave, once the inaudible sound waves from the transducers interfere, the waves create a focal point that is perceived as a solid object. A camera tracks the position of a user’s hand and shifts the transducers’ output to move the focus around with the movement of the hand. The result: a feeling of tracing the virtual object’s surface or edges in mid-air with one’s bare hands.

The system is “the first of its kind,” says Stephen Brewster, a haptics researcher at the University of Glasgow. “You can feel it with both hands, rather than having just a single point of contact, and multiple people can use it at the same time.”

Iwamoto’s team is currently adjusting how the transducers are driven in order to create realistic textures and shapes. The team plans to combine their system with 3D modeling software and video games, and has received proposals from several entertainment companies.

---

**In 2005, Yahoo! was engaged in four patent-related lawsuits; by 2008, the number had swollen to 22. All are plaintiff-driven cases.**
an invention. Yet it was a setback for the pharmaceutical industry, which often seeks new patents for the combination of one drug with another.

**Enter Big Pharma**

Because drugs typically contain only one or two patented compounds, and the pharmaceutical industry relies far more heavily on patents than any tech company does, the battle over patent reform is frequently depicted as pitting IT against Big Pharma. But the IT industry itself is far from unified. Companies like Qualcomm, Tessera, and Rambus are highly dependent on patent revenue, and are therefore deeply suspicious of the reforms proposed by their peers. Another source of disagreement is the apportionment of patent-related damages, which many IT heavyweights complain have recently ballooned in a manner that’s disproportionate to the value of infringed inventions.

“If you can’t cure the proliferation of questionable patents, you try to reduce their ramifications,” says Robert Barr, executive director of the Berkeley Center for Law & Technology.

An injured patent holder is entitled to pursue several different remedies. Injunctive relief prohibits the defendant from continuing to use or sell the infringed invention. Once common, injunctions have become more difficult to obtain since 2006 when the Supreme Court ruled that they could not automatically be issued to non-practicing entities. Lost profits damages, which are difficult to prove and expensive to analyze, have also fallen out of favor. Most plaintiffs thus opt to seek “reasonable royalties” from the defendant. As a matter of convenience, these royalties are often calculated as a percentage of overall product sales. This angers many in the high tech arena, who claim the calculations don’t correspond to the specific value of an infringed patent. Consider the earlier example of a microchip, with its thousands of patented components. If a company were to sue for the infringement of a single component and win, it could ask for damages representing a percentage of the sales of the entire chip.

The Patent Reform Act seeks to redefine “reasonable royalties” to reflect only the economic value of a patent’s “specific contribution over prior art” or, as Senator Leahy described it, “the truly new ‘thing’ that the patent reflects.” It was one of the bill’s most hotly contested provisions, drawing criticism from both the pharmaceutical industry and certain IT segments. Their chief complaint: the value of a product may not be separable from the value of an individual component.

“It suggests that the whole is divorced from its parts,” asserts Brad Ditty, associate general patent counsel at InterDigital Communications. “And it artificially lowers the value of a patent.” Ditty and his peers prefer the flexibility of the current system, and they see no need for reform. Nor do they believe an imbalance exists. “There’s this notion that we’re currently in the midst of a crisis as far as damage awards are concerned,” says Ditty. “We just don’t see it.”

One proposal that remains controversial with the tech community is the Patent Reform Act’s third major provision, which would change the way patents are granted from a first-to-invent to a first-to-file system. (Although some individual inventors have complained that this would put them at a disadvantage relative to larger companies, studies have shown that the first person to file for a patent is almost always the first to invent.) Such a change would bring the U.S. system in line with the rest of the world, and would streamline the approval process by eliminating messy debates about who first had an idea. In fact, it is one of the bill’s few provisions that the pharmaceutical industry also supports, and industry insiders regret that IT companies have not been able to use it to greater advantage to score concessions on other points.

At press time, there was no schedule for the Patent Reform Act’s return. Senator Leahy has said he remains committed to patent reform, but a growing consensus surmises that supporters of the legislation may need to wait until 2009, when there is a new Congress, a new President, and a new head of the USPTO. In the meantime, the battle will continue to be waged, at great expense, in the courts.

---

**Quantum Computing**

**Alexei Kitaev Wins MacArthur “Genius” Award**

Alexei Kitaev, a professor of theoretical physics and computer science at the California Institute of Technology, is one of 25 recipients of a MacArthur Foundation $500,000 “genius” award. As a 2008 MacArthur Fellow, Kitaev will receive $100,000 a year for five years, with no strings attached.

Kitaev said in a statement that he was “very surprised” when he received a call from MacArthur Fellows Program director Daniel Socolow informing him of his selection.

“I didn’t know what the award was at first,” said Kitaev. “But then I looked up the names of people who have previously received a MacArthur award and saw that they are very good scientists. I am excited and honored to be in the same group with them.”

A physicist, Kitaev was cited by the MacArthur Foundation for his work in the nature of quantum systems and their implications for creating practical uses, such as quantum computers. “Though his work is focused mainly at the conceptual level, he also participates in efforts to develop working quantum computers,” the foundation noted. “Through his deep insights into the fundamental nature of quantum physics, Kitaev reveals a rich picture of this unfamiliar world, bringing us closer to the realization of the full potential of quantum computing.”

Kitaev conducted his undergraduate and graduate work in Russia, and came to Caltech as a visiting associate and lecturer in 1998 and was named a professor of theoretical physics and computer science in 2002.

---

Leah Hoffmann is a Brooklyn, NY-based science and technology writer.
Legally Speaking

Why Do Software Startups Patent (or Not)?

Assessing the controversial results of a recent empirical study of the role of intellectual property in software startups.

Two-thirds of the approximately 700 software entrepreneurs who participated in the 2008 Berkeley Patent Survey report that they neither have nor are seeking patents for innovations embodied in their products and services. These entrepreneurs rate patents as the least important mechanism among seven options for attaining competitive advantage. Even software startups that hold patents regard them as providing only a slight incentive to innovate.

These are three of the most striking findings from a recently published article, “High Technology Entrepreneurs and the Patent System: Results of the 2008 Berkeley Patent Survey.” After providing some background about the survey, I will discuss some key findings about how software startup firms use and are affected by the patent system.

While the three findings highlighted above might seem to support a software patent abolitionist position, it is significant that one-third of the software entrepreneur respondents reported having or seeking patents, and that they perceive patents to be important to persons or firms from whom they hope to obtain financing.

Some Background on the Survey

More than 1,300 high-technology entrepreneurs in the software, biotechnology, medical devices, and computer hardware fields completed the Berkeley Patent Survey. All of these firms were no more than 10 years old before the survey was conducted. We drew our sample from a general population of high-tech firms registered with Dun & Bradstreet (D&B) and from the VentureXpert (VX) database that has a rich data set on venture-backed startups. (Just over 500 of the survey software respondents were D&B firms; just under 200 respondents were VX firms.)

Eighty percent of the software respondents were either the CEOs or CTOs of their firms, and most had experience in previous startups. The average software firm had 58 employees, half of whom were engineers. Between 10%–15% of the software startup respondents among the D&B respondents were venture-backed firms. Among the software respondents, only 2% had experienced an initial public offering (IPO), while 9% had been acquired by another firm.

Our interest in conducting this survey arose because high-technology entrepreneurs have contributed significantly to economic growth in recent decades. They build firms that create new products, services, organizations, and opportunities for complementary economic activities. We were curious to know the extent to which high-tech
startups were utilizing the patent system, as well as to learn their reasons for choosing to avail themselves of the patent system—or not.

The basic economic principle underlying the patent system is that technology innovations are often expensive, time-consuming, and risky to develop, although once developed, these innovations are often inexpensive and easy to copy; in the absence of intellectual property rights (IPRs), innovative high-tech firms may have insufficient incentives to invest in innovation insofar as they cannot recoup their research and development (R&D) expenses and justify further investments in innovation because of cheap copies that undermine the firms’ recoupment strategy.

Although this economic principle applies to all companies, early-stage technology firms might, we conjectured, be more sensitive to IPRs than more mature firms. The former often lack various kinds of complementary assets (such as well-defined marketing channels and access to cheap credit) that the latter are more likely to enjoy. We decided it would be worthwhile to test this conjecture empirically. With generous funding from the Ewing Marion Kauffman Foundation, three colleagues and I designed and carried out the survey and have begun analyzing the results.

**Why Startups Patent**

The most important reasons for seeking patents, as reported by software executives who responded to the Berkeley Patent Survey, were: to prevent competitors from copying the innovation (2.3 on a 4 point scale, where 2 was moderately important), to enhance the firms’ reputation (2.2), and to secure investment and improve the likelihood of an IPO (1.96 and 1.97 respectively).

The importance of patents to investors was also evident from survey data showing striking differences in the rate of patenting among the VX and the D&B software companies.

Three-quarters of the D&B firms had no patents and were not seeking them. Because the D&B firms are, we believe, fairly typical of the population of software startup firms in the U.S., their responses may well be representative of patenting rates among software startups generally. It is, in fact, possible that the overall rate of software startup patenting is lower than this given that patent-holders may have been more likely than non-patent-holders to take time to fill out a Berkeley Patent Survey.

In striking contrast to the D&B respondents, over two-thirds of the VX software startup respondents in the sample, all venture-backed, had or were seeking patents. We cannot say why these venture-backed firms were more likely to seek patents than other firms. Perhaps venture capitalists (VCs) are urging firms they fund to seek patents; or VCs may be choosing to fund the development of software technologies that VCs think are more amenable to patenting.

Interestingly, the rate of patenting did not vary by the age of the firm (that is, older firms did not patent more than younger firms).

**Why Forego Patenting?**

The survey asked two questions about decisions to forgo patenting: For the last innovation for which the firm chose not to seek a patent, what factors influenced this decision, and what was the most important factor in the decision?

The costs of obtaining and of enforcing patents emerged as the first and second most frequent explanation. Twenty-eight percent of the software startup executives reported that the costs of obtaining patents had been the most important factor in this decision, and 12% said that the costs of enforcing patents was the most important factor. (They reported that average cost of getting a software patent was just under $30,000.)

---

**Calendar of Events**

November 17–19
Asian Internet Engineering Conference, Bangkok, Thailand, Contact: Kanchanasut Kanchana, Email: kk@cs.aist.ac.th

November 17–19
Advances in Computer Entertainment Technology Conference, Taipei, Taiwan, Contact: Duh Henry B.L., Email: eledbl@nus.edu.sg

November 22–23
Conference on Decision and Game Theory for Security, Berlin, Germany, Contact: Tansu Alpcan, Email: alpcan@sec.t-labs.tu-berlin.de

November 22–24
The 17th ACM Symposium on Virtual Reality Software and Technology, Hong Kong, Contact: George Baciu, Email: csgeorge@comp.polyu.edu.hk

December 1–3
9th International Conference on Mobile and Ubiquitous Multimedia, Limassol, Cyprus, Contact: Angelides Marios, Email: marios.angelides@brunel.ac.uk

December 4–8
The 43rd Annual IEEE/ACM International Symposium on Microarchitecture, Atlanta, GA, Sponsored: SIGMICRO, Contact: Sudhakar Yalamanchili, Email: sudhak@ece.gatech.edu

December 5–8
Winter Simulation Conference, Baltimore, Maryland, Sponsored: SIGSIM, Contact: Joe Hugan, Email: jhugan@gmail.com

December 12–13
Virtual Reality Continuum and its Applications in Industry, Seoul, Republic of Korea, Sponsored: SIGGRAPH, Contact: Hyunseung Yang, Email: hyang@cs.kaist.ac.kr

---
Ease of inventing around the innovation and satisfaction with secrecy also influenced software startup decisions not to seek patents, although only rarely were these factors considered the most important.

Intriguingly, more than 40% of the software respondents cited the unpatentability of the invention as a factor in decisions to forego patenting. Almost a quarter of them rated this as the most important factor. Indeed, unpatentability ranked just behind costs of obtaining patents as the most frequently cited “most important factor” for not seeking patents.

It is difficult to know what to make of the unpatentability finding. One explanation may be that the software respondents believed that patent standards of novelty, non-obviousness, and the like are so rigorous that their innovation might not have satisfied patent requirements. Yet, because the patentability of software innovations has been contentious for decades, it may also be that a significant number of these entrepreneurs have philosophical or practical objections to patents in their field.

**How Important Are Patents to Competitive Advantage?**

One of the most striking findings of our study is that software firms ranked patentability of the invention as a factor in decisions to forego patenting. Almost a quarter of them rated this as the most important factor. Indeed, unpatentability ranked just behind costs of obtaining patents as the most frequently cited “most important factor” for not seeking patents.

It is difficult to know what to make of the unpatentability finding. One explanation may be that the software respondents believed that patent standards of novelty, non-obviousness, and the like are so rigorous that their innovation might not have satisfied patent requirements. Yet, because the patentability of software innovations has been contentious for decades, it may also be that a significant number of these entrepreneurs have philosophical or practical objections to patents in their field.

**What Incentive Effects Do Patents Have?**

The Berkeley Patent survey asked startup executives to rate the incentive effects of patents on a scale, where 0 = no incentive, 1 = weak incentive, 2 = moderate incentive, and 3 = strong incentive, for engaging in four types of innovation: (1) inventing new products, processes, or services, (2) conducting initial R&D, (3) creating internal tools or processes, and (4) undertaking the risks and costs of commercializing the innovation.

We were surprised to discover the software respondents reported that patents provide only weak incentives for engaging in core activities, such as invention of new products (0.96) and commercialization (0.93). By contrast, biotech and medical device firms reported just above 2 (moderate incentives) for these same questions.

Interestingly, the results did not change significantly when considering only responses from software entrepreneurs whose firms hold at least one patent or application. Even patent-holding software entrepreneurs reported that patents provide just above a weak incentive for engaging in these innovation-related activities.

**Resolving a Paradox**

If patents provide only weak incentives for investing in innovation among software startups, why did two-thirds of the VX respondents and at least one-quarter of the D&B respondents seeking patents? The answer may lie in the perception among software entrepreneurs that patents may be important to potential funders, such as VCs, angel investors, other firms, commercial banks, and friends and family. Sixty percent of software startup respondents who had negotiated with VCs reported that they perceived VC decisions about whether to make the investments to be affected by patents. Between 40% and 50% of the software respondents reported that patents were important to other types of investors, such as angels, investment banks, and other companies.

**Controversy Over Survey Findings**

It is an article of faith among many IP lawyers that patents provide significant incentives for firms to engage in R&D and develop new products. Most would also expect, as we did, that high-tech startup companies would regard patents as more important as an inducement to innovation than large firms, given that the latter have lots of other assets for achieving and maintaining success in the marketplace.

Anecdotes highlighting the importance of patents to high-tech entrepreneurs are relatively easy to find. Because data from the Berkeley Patent Survey suggests that software entrepreneurs regard patents as quite unimportant, the reaction of some prominent patent lawyers to our article about the survey has been sharply negative. We believe, however, that our analysis is sound and these critiques are off-base. We encourage readers to read the full article and make their own judgments.

**Future Research**

Over the next several years, the co-authors of the Berkeley Patent Survey article expect to analyze further data from this survey and to report new findings. We will look more closely, for example, at differences in patenting rates among those in different sectors of the software industry and differences between patent holders and non-patent holders. We know already that product innovators seek patents more often than process innovators.

The findings reported here suggest that software entrepreneurs do not find persuasive the canonical story that patents provide strong incentives to engage in technology innovation. These executives regard first-mover advantage and complementary assets as more important than IPRs in conferring competitive advantage upon their firms. Moreover, among IPRs, copyrights and trademarks are perceived to be more important than patents. Still, about one-third of the software entrepreneur respondents reported having or seeking patents, and their perception that their investors care about patents seems to be a key factor in decisions to obtain patents.

Reference


Pamela Samuelson (pam@law.berkeley.edu) is the Richard M. Sherman Distinguished Professor of Law and Information at the University of California, Berkeley.

Copyright held by author.
If one was forecasting the future of software today, it is likely that open source software (OSS) would figure prominently in most projections. Indeed, open source zealots might expect to see OSS everywhere, with “innovation networks” abounding, Microsoft humbled, and Linux on every desktop. Personally, I wouldn’t bet on it.

Historians are cautious about forecasting the future, with good reason. They know that when technical experts gaze into the crystal ball, they usually extrapolate well but fail to spot those discontinuities that can transform a technology. One such attempt at futurology was the book *The Future of Software*, published in 1995. The book included contributions from leading experts in the field. They correctly extrapolated that PCs would become more powerful, numerous, pervasive, and software would proliferate to fill the applications vacuum. That was correct to a point, but their collective take on new software development methods and technologies was wide of the mark. One contributor forecast that visual programming by ordinary users would herald the “fall of software’s aristocracy.” Another predicted the maturing of the software factory, by which our “craft industry” would be transformed “toward Ford-style mass production.” Another contributor expected to see stunning advances in natural language interfaces. What no contributor foresaw, or even mentioned, was the impact of open source software and development techniques. At the very moment they were making their projections, Linux was under their nose but they could not see it.

The idea of open source software goes back to the very dawn of computing, when the mainframe computer was getting established in the early 1950s. At that time, and for many years after, IBM and the other computer manufacturers gave their software away for free—software was seen largely as a marketing initiative that made their hardware more saleable. Software was supplied in both source and object code form because some people found the source code useful and there was no reason not to let them have it. Where manufacturers’ provision fell short, cooperative user groups, such as IBM’s SHARE, coordinated the writing and free distribution of programs. When it came to applications, computer users wrote their own or hired a “software contractor,” such as the Computer Sciences Corporation or Electronic Data Systems, to write software for them.

There was a radical transformation in the software world in 1964, with the
launch of IBM’s System/360 computer. The 360 created, for the first time, a standard computer platform, and it massively expanded the computer population, particularly in medium-sized businesses. Most of the new computer owners did not have the resources to hire a staff of programmers or to buy the services of a software contractor. There was thus an applications vacuum filled by the first software product firms. These firms wrote programs for specific industries (such as the insurance or construction industries), or for generic, cross-industry functions (such as payroll or stock control). The sales of individual software products were quite modest: if a product had 100 or so customers it was considered quite successful. Software product prices were high, typically $50,000 upward. This was not only because of the low sales volume, but because software writing was very capital intensive. The only way to run a software business was to hire a team of programmers plus a mainframe computer and put them to work. This cost at least $1 million a year (closer to $10 million in today’s currency).

The first software products were usually supplied in both source code and object code. This was necessary because customizing software was a little-understood technology and most users configured their application software by modifying the source and recompiling it. Software-product companies were, naturally, concerned about disclosing source code, because if it fell into the hands of a competitor it would make it easy for them to produce a competing product. In a somewhat uneasy compromise, paying customers received a copy of the source code but were bound by the license terms with a trade secrecy clause requiring them not to disclose the source code or documentation to third parties.

The advent of personal computers, which occurred during the late 1970s, gave rise to a new software industry that rewrote the rules for making and selling software. The cost of computer power plummeted, the computer population soared, and the number of software firms increased exponentially. However, although the hardware-cost barrier to software making had been lowered, code development still needed a disciplined environment of salaried programmers who worked office hours in the same physical location. Although computer networks existed in the 1980s, they were slow and impractical—software development remained a same-time, same-place, collaborative activity. PC software products were comparatively inexpensive (usually less than $500), but this was only because the sales volume was high compared with mainframe software. Software writing remained an expensive, highly capitalized activity.

In the new PC environment, with thousands of software companies and millions of users, it was no longer feasible for software companies to supply their source code to users, or their products would be rapidly duplicated. Firms such as Microsoft, Lotus, and WordPerfect had invested hundreds of millions of dollars in software development; disclosing their software would have been akin to giving away the family jewels. Of course, software had some legal protection through copyright laws, but this did not protect the data structures and algorithms that would have been exposed by access to the source code. By the mid-1980s source code disclosure had almost completely ceased—in 1983, IBM was one of the last major companies to stop disclosing source code in its so-called OCO (object-code only) policy. Competitors and users alike objected to the OCO policy, but IBM was resolute and was doing no more or less than the rest of the industry. By the mid-1980s, trade secrecy was endemic in the software products industry.

The ascendency of the Internet in the early 1990s began another radical transformation of software development. Inexpensive network access removed the idea of open source software goes back to the very dawn of computing, when the mainframe computer was getting established.
The first software products were usually supplied in both source code and object code.

Martin Campbell-Kelly (M.Campbell-Kelly@warwick.ac.uk) is a professor in the Department of Computer Science at the University of Warwick, where he specializes in the history of computing.
In his discourse comparing various economic systems, Schumpeter [5] declares it is new products, new markets, and new forms of production and distribution that impel the creative destruction engine of free enterprise. Entrepreneurs strategically weave an organizational design of customer value, product offering, and production and distribution technologies that enables them to compete with, and often displace, existing organizations. Customers decide whether to accept the new firm’s offerings based on their perceptions of value.

Open source software (OSS) appears to be creative destruction in action: all three of the components that fuel the destructive fire are evident. There is an abundance of new and innovative products emerging from the OSS community. The zero-cost licensing structure of most open
source projects has opened up the acceptance of these products into a number of previously untapped markets. The Internet has created an environment in which software distribution costs are approaching zero. Products freely and rapidly flow across borders. There is no packaging—no shelf-space requirements. OSS is not confined to one economic system. It overcomes both the tyranny of distance [1] and oppression of borders. OSS also espouses new methods of software production that utilize the public as the production mechanism and allows for coordinated, location-agnostic access to the raw materials for these new products.

To assess if OSS has the potential to revolutionize the development and distribution of software, we must first understand how the entities involved in the development of such software are organized. Because different business models are not equal in their capacity to create value, we must analyze each particular blend of customer, product, and production and distribution mechanism. After presenting five models underlying the software development business, we focus our attention on the one that has perhaps the most disruptive potential.

**The Business Models**

We distinguish five models of software production or distribution: proprietary, open community, corporate distribution, sponsored OSS, and second-generation OSS. Whereas the first two constitute the extremes of the closed-open continuum, the other three are hybrids of closed and open models.

**Proprietary and Open Communities.** Proprietary and open communities both have their origins in the early days of computing, when some people freely exchanged code while others recognized there were customers for their programs and accordingly sold executable versions of their products while carefully securing the source.

The proprietary model has dominated the marketplace for decades. Firms employ programmers to develop software and customers purchase it. The code is considered a major intellectual resource, and traditional software firms protect their code from outside eyes by erecting physical and legal firewalls between their code and the outside world. Proprietary firms rely heavily on both copyright law—to ensure that “leaked” source code cannot legally be used in a competing product—and patent law—to protect their intellectual property from duplication. While the code is most often sold for license fees, proprietary firms can and do distribute their products as freeware. It is not the price that distinguishes proprietary software but rather the public’s inability to view and modify the source code.

At the other extreme is the open community model, which involves the development and support of software by volunteers with limited or no commercial interest. This model dominates the OSS movement in terms of number of projects. Many of them can be located through large, Internet-based project management and source code repositories such as SourceForge, which hosts over 170,000 software projects. While the majority of these projects involve only one or two developers and have a small number of users, many have a vast base of both developers and users and have produced products that provide unique functionality or offer compelling alternatives to commercial products.

**Corporate Distribution.** Based on the high level of adoption for many OSS products, it seems apparent that quality products are being produced through the open community model. However, some entrepreneurs recognize that identifying appropriate products, interacting with open communities for support, and developing the required support skills can be challenging for many potential OSS customers. As a result, firms, such as RedHat, SpikeSource, and OpenOSX, have emerged to create value (and generate revenue) by identifying best-of-breed OSS projects, improving distribution methods for these products, and providing complementary services in order to make these OSS products more accessible to a broader market.

**Sponsored Open Source.** Corporations and foundations sponsor some OSS projects. For example, the Apache Software Foundation fosters the development of the Apache server and over 50 other OSS projects. Some corporate sponsors directly contribute development resources to OSS projects. IBM is a high-profile example of a corporation contributing developers to Apache’s Web server. In some cases, sponsored OSS projects have been initiated by corporations releasing previously closed code and encouraging their employees to continue to work on the now open project. Eclipse, an integrated software development environment, was released as OSS by IBM, whose developers are still primary contributors to the project.

**Second-Generation Open Source.** Second-genera-

---

1As of late February 2008, SourceForge hosted 170,539 projects.
tion open source (OSSg2)—also known as professional open source—firms are essentially a hybrid between corporate distribution and sponsored OSS. As with the corporate-distribution model, OSSg2 companies typically generate the bulk of their revenues by providing complementary services around their products [3]. Like sponsored projects, OSSg2 firms provide the majority of the development resources required to create and maintain their products. However, unlike most corporate-distribution companies, OSSg2 firms generally do not sell licenses for their products, and unlike most sponsored projects, OSSg2 firms typically own or tightly control the software code and can exploit their intimate knowledge of the code to provide higher-quality service than could potential competing service providers. As the leading OSSg2 firms (including JBoss, MySQL, Trolltech, and Sleepycat) are privately held or have been acquired, we do not have data on their profitability. However, based on interviews with the CEOs of the four firms mentioned here, it appears they are cash-flow positive while growing rapidly.

We contend that OSSg2 firms have a very promising business model that could emerge as a dominant model for OSS development in the coming years. Here, we examine four leading OSSg2 companies, highlighting three important benefits of their business models.

**EVALUATING THE OSSG2 BUSINESS MODEL**

We studied four companies that are among the OSSg2 leaders. In order of business longevity, these are Trolltech, MySQL, Sleepycat, and JBoss. Three important characteristics of the OSSg2 model exhibited by these firms that lead to specific benefits are: accountability (and the benefit of reduced liability problems); talent base (and the associated benefits for code quality and support); and ecosystem (and the associated benefits of trialability and quality assurance). These benefits should improve the value proposition to customers for OSSg2 products and are central to our contention that OSSg2 is a threat to traditional software firms. To quote Marc Fleury, former CEO of JBoss: “We (OSSg2 companies) are proving that professional open source can do it better and faster and cheaper than our traditional competitors.” How the OSSg2 model addresses three specific strategic risks is discussed later in this article.

**OSSG2 Leaders.** Trolltech was founded in Norway in 1994 and currently has more than 4,400 customers. It manages two software products: Qt, a cross-

Because different business models are not equal in their capacity to create value, we must analyze each particular blend of customer, product, and production and distribution mechanism.

---

2 As discussed here, some OSSg2 firms offer a proprietary license for organizations not wishing to comply with the terms of the OSS licensing model.

3 JBoss was acquired by RedHat, Inc. in April 2006.

4 Sleepycat was acquired by Oracle, Inc. in February 2006.

5 All uncited quotes are based on personal interviews.
augment the original source code and do not wish to release the modifications under an OSS license, they must buy a commercial license. Mike Olson, former CEO of Sleepycat, acknowledges that the dual licensing strategy is a “great judo trick for competing with proprietary vendors.” JBoss is based on a Lesser General Public License (LGPL) license and only receives revenues from services including software support, training, and consulting. A second difference is that, whereas Trolltech, MySQL, and Sleepycat own the source code underlying their products (which allows them to offer a dual-licensing scheme), JBoss does not own the code of the software products it services. Nevertheless, as an OSSg2 firm, it controls the code more tightly than companies based on the other OSS business models. For example, the extent of its contribution to its three leading products is 85% for JBoss AS, 95% for Hibernate, and 60% for Tomcat [4].

OSSg2 Characteristics Improve Customer Value. Three key traits of OSSg2 firms provide specific benefits to their customers and can thus improve their value proposition.

Accountability. All four OSSg2 companies we studied indemnify their paying customers from any legal liability associated with their products (potential patent or copyright infringements). The indemnity provision provides a necessary level of security for potential adopters of OSSg2 products who are still apprehensive about OSS. This accountability may do more than just bring OSSg2 companies to parity with proprietary software vendors. Sleepycat’s former CEO, Mike Olson, asserts the risks of patent infringements and copyright problems are lower for an OSSg2 company: “Anyone that wants to can look at my software. If there was a claim pending, if I had stolen something, it is overwhelming likely that it would have been seen by now. […] No proprietary vendor’s customer has that degree of assurance. If there has been intellectual property misappropriated in that product, it is a secret and it may still be lurking” [7].

From Olson’s perspective, OSSg2 companies offer greater peace of mind than proprietary firms, because patent or copyright infringements should be uncovered relatively early—reducing the impact the copyright infringements may have on adopting customers’ systems.

Talent base. Each OSSg2 company retains talented coders, wherever they are located, to maintain and support its software products. Indeed, all four OSSg2 leaders insist they recruit from among the world’s best and most productive programmers, drawing not only from the immense pool of first-generation OSS contributors, but also from a growing collection of emerging talent. MySQL and JBoss allow their programmers to live wherever they wish. In addition, recruitment is different. Typically, those employed by OSSg2 companies have an established record of contributing code and identifying bugs as volunteers prior to their hiring. They have demonstrated their understanding of the code base and their ability to fit within the OSS development culture. This is an important competitive advantage because it means OSSg2 companies reduce hiring risks without significant up-front recruiting and training costs.

Trolltech’s employees (approximately 230) come from more than 20 different countries; they were recruited almost exclusively through the OSS community. Trolltech has learned that great developers want to work with each other. When asked about the criteria for hiring, CEO Havard Nord emphasized that what really counts is “code, code, and code…merits…formal education is less important.” Trolltech’s employees are the company’s most valued assets. The founders and the employees—owning two-thirds of the shares—control Trolltech. Outside investors have majority ownership and control of many software companies. Only in rare cases does Trolltech get outside contributions for its products. When this happens, either the submitting contributor is hired (if coding quality and knowledge of the product have been demonstrated) or the firm gets ownership of the code. As a result, Nord claims Troll-
tech’s customer base is “extremely happy with the code,” thanks to the company’s careful approach to recruiting and retaining its high-skill employee base. Fleury insists that employees are JBoss’s greatest asset as well: “I treat my elite developers like royalty. I overpay them. I cover my lead developers in stock. Many of them walk around with an executive package, which is rarely the case, if ever, in traditional software companies.”

This focus on hiring the best programmers results in a quality of code that is at least commensurate with that of proprietary development while maintaining the benefit of “mass innovation” [4] shared by all OSS products. Similarly, the support and education offered by OSSg2 companies meet industry expectations because of the quality of the personnel.

**Ecosystem.** There is an encompassing ecosystem that evolves around OSSg2 companies that typically includes all the entities that gain from the OSSg2 companies’ presence in the market (support services, authors, educators, publishers, partners, user communities, and so forth). This translates into multiple Web sites, email lists, newsgroups, conferences, and published materials providing up-to-date information about OSSg2 products and their applications. OSSg2 companies typically benefit greatly from their ecosystem without much strain on their resources. Mårten Mickos, CEO of MySQL, emphasizes that MySQL tries, with minimum involvement, to make its ecosystem thrive: “We try to be open about our intentions so it’s easy for others to plan their business and their life around us. We try to move the obstacles of getting our product, distributing our product, using our product. […] We just make sure the friction is as low as possible.”

Potential OSSg2 customers can download and test a complete software product extensively before making an adoption decision. Because the ecosystem provides an effective pre-sales support apparatus, potential customers receive a significant advantage in the form of trialability that is limited neither in time nor functionality.

The ecosystem can also provide for an efficient, external quality-assurance mechanism above and beyond what may be carried out in-house, as Mickos points out: “When we release a new version, within 24 hours 35,000 people have downloaded and tested it. That’s fantastic. Not even Microsoft has 35,000 QA engineers. […] Just based on statistics, we know that there are enough people out there who certainly test all relevant features, without our specific instruction” [7].

The OSSg2 model thus has a significant advantage in leveraging an important ecosystem that is willing to work on its behalf.

**How OSSg2 Addresses Risks.** OSSg2 has an adroit answer to dealing with the major risks facing all software firms. Every firm faces three strategic risks: demand, efficiency, and innovation [2].

**Demand risk and pricing strategy.** Wal-Mart and Dell have altered the structure of the retailing and computer industries through their low-cost strategies. Similarly, OSSg2 firms push the cost of software acquisition to the lower limit. Assuming requirements are met by an OSSg2 product, cost-driven IS departments will be attracted by zero acquisition costs. Extensive trialability, discussed earlier, also contributes to mitigate demand risk. For OSSg2 firms, as with both Wal-Mart and Dell, revenue losses from low-cost strategies are largely offset by increased operational efficiencies.

**Efficiency risk and the Internet.** OSSg2 firms gain from efficiencies associated with their Internet-based infrastructures. Many employees work remotely, software is downloaded rather than packaged and distributed physically, and high trialability obviates many traditional marketing costs. Consequently, OSSg2 firms tend to have a lower cost structure than traditional firms and thus enjoy efficiency differentials over proprietary software competitors.

**Innovation risk and open source.** When code is open, many coders can inspect it, and faults often will be detected more rapidly than when only a handful review it. Furthermore, those who can see the code can suggest improvements and submit code changes. As with all OSS communities, the developers and supporting community members for OSSg2 projects are drawn from all areas of the world, an immense talent pool from which OSSg2 community members can be recruited on the basis of talent and contribution, unfettered by physical location. This ready supply of programmers ensures innovative ideas can be contributed to the OSSg2 community from both traditional sources and sources previously untapped by traditional software firms. This phenomenon directly attacks innovation risk.

**CONCLUSION**

The open source movement is challenging the status quo in the software marketplace. Open source programs have moved beyond the desktops of code hackers and are now in production in a growing number of corporate IS departments. OSS is ampli-
fying the demand, efficiency, and innovation risks that traditional software organizations face and is driving a period of creative destruction that has the potential to permanently alter the competitive landscape within the software industry. OSSg2 firms offer a significant customer value proposition and have effective strategies that should aid their prospects for long-term survivability. However, during this period of creative destruction, we also recognize that the market is constantly changing, traditional firms are experimenting with adjustments to their strategies to address the stresses that OSSg2 is placing on their business models, and new models are emerging that will blur the lines between the categories we have outlined in this article. This makes the business of open source both extremely fascinating and highly consequential.

REFERENCES

RICHARD T. WATSON (r watson@terry.uga.edu) is the J. Rex Fuqua Distinguished Chair for Internet Strategy in the Terry College of Business at the University of Georgia, Athens. MARIE-CLAUDE BOUDREAU (mcboudre@terry.uga.edu) is an associate professor in the Terry College of Business at the University of Georgia, Athens, GA. PAUL T. YORK (p tyork@uga.edu) is a Ph.D. student in the Terry College of Business at the University of Georgia, Athens, GA. MARTINA E. GREINER (m greiner@uga.edu) is a Ph.D. student in the Terry College of Business at the University of Georgia, Athens, GA. DONALD WYNN, JR. (wynn dona@notes.udayton.edu) is an assistant professor in the Department of Management Information Systems, Decision Sciences and Operations Management at the University of Dayton, OH.
Perhaps the business case for adopting open source software is an easy sell. After all, the software is free, and can be simply downloaded from the Internet and installed or customized as needed. Organizations interested in reducing the licensing fees of proprietary software, while also avoiding the penalties and legal liabilities associated with their illegal use, can definitely consider open source software a plausible alternative. However, less obvious than the cost savings but equally important are the barriers (“hidden costs”) of adopting open source software.

Open source software has created considerable excitement in the business world over the last decade. These applications, designed by groups of volunteer software developers, have the potential to break the current dominance of proprietary software and restrictive licenses for many business applications, reduce software development time and improve software quality, and most importantly, bring much needed software applications within the reach of individuals and small businesses, who cannot otherwise afford such software. Further, unlike proprietary software, open source software applications make their source code available for free, which can be customized to fit the unique needs of specific organizations.

Many organizations have caught on to open source software and realized significant cost savings in technology expenditure as a result. For instance, Cendant Travel Distribution Services replaced a $100 million mainframe system with a $2.5 million system running on 144 Linux servers. Amazon.com cut its technology expenditure from $71 million to $54 million by switching to open source applications. Sabre Holdings saved tens of millions of dollars by adopting MySQL, an open source database product. Though the basic open source software is free, the prospect of paid ancillary products and services such as hardware and consulting has motivated many erstwhile proprietary technology vendors such as Hewlett-Packard, IBM, and Sun Microsystems to embrace open source software and offer value-added services based on such software.

Table 1 shows an estimated range of the current global market share of several of today’s open source software applications. This table shows that though the open source market is large and growing for some application domains such as Web server (such as Apache), server operating systems (such as Linux Server), database server (such as MySQL), electronic mail client (such as Sendmail), and Internet browser (such as Firefox), it is lagging behind its proprietary counterparts in other domains such as client operating systems (such as Linux Workstation), office productivity software (such as OpenOffice), and enterprise resource planning (ERP) systems. This pattern suggests that there may be significant barriers to open source software adoption among some sectors of the user populations.
It is widely believed that proprietary software vendors often use fear, uncertainty and doubt to undermine and cut the market potential of their open source competitors. The objective of this paper is to reduce that uncertainty via a candid discussion of the barriers confronting open source software adoption and potential remedies to those barriers. These barriers and their remedies, summarized in Table 2, are discussed in detail.

**Knowledge Barriers**

Given all of the above benefits of open source software, one is left with the question: why do some organizations still spend money on proprietary software? In some cases, the answer is simply that they do not know that an open source application relevant to their business needs exists. For instance, Compiere, an open source ERP systems for small-to-medium sized businesses, may be reasonable alternatives to expensive ERP systems from SAP and Oracle, but is largely unknown among the adopter community. Equally unknown are open source business intelligence and analysis software such as Jasper Reports and Pentaho. In other cases, managers may be aware of open source applications, but may lack the knowledge required to implement and use it. Further, they may be unaware of the availability of support services or consultants who can assist with open source software implementation or the range of services offered. Unlike their proprietary counterparts, open source software projects are often run by volunteer organizations without marketing or advertising budgets, who have fewer means or resources of reaching or informing their potential adopter base.

What can organizations do to overcome this knowledge gap? Given the rapid rate of changes in business practices and available technologies to support them, it is practically impossible for organizations outside of the software domain to keep track of all available software products relevant to their operations. However, one convenient Web site they can occasionally visit to search for and download relevant applications is SourceForge.net. This Web site maintains a “living” archive of over 131,000 open source applications, along with their latest software patches and updates, covering a broad spectrum of software from specialized technical software, such as operating systems and middleware, to business applications software, such as ERP systems and point of sale systems.

Adopting new software applications is also associated with a steep learning curve, and more so in the case of open source software, given the relative lack of a support network to assist with their implementation, deployment and support. These applications vary significantly in terms of their usability or user-friendliness, from command lines to graphical user interfaces. Further, since more open source applications are newer systems relative to their proprietary counterparts, little legacy integration is needed to customize it and use it, or business knowledge needed to implement it.

<table>
<thead>
<tr>
<th>Barrier</th>
<th>Description</th>
<th>Proposed Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge barriers</td>
<td>Lack of awareness of software availability or relevance, technical knowledge needed to implement and use it, or business knowledge needed to customize it</td>
<td>Monitor open source archives, train internal staff, outsource system implementation and maintenance</td>
</tr>
<tr>
<td>Legacy integration</td>
<td>Lack of ability to connect to existing legacy systems</td>
<td>Use middleware solutions</td>
</tr>
<tr>
<td>Forking</td>
<td>Open source software is written by different groups and may not interoperate with each other or to other applications</td>
<td>Self-resolving with the development of self-managed standards bodies</td>
</tr>
<tr>
<td>Sunk costs</td>
<td>Prior investments in proprietary software</td>
<td>Consider open source in areas without proprietary software, compare future cost streams of maintaining proprietary versus open source software</td>
</tr>
<tr>
<td>Technological immaturity</td>
<td>Non-professional development with considerable variation in available support.</td>
<td>Open source software maturity models and independently evaluate case studies.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Open Source Software</th>
<th>Type of Application</th>
<th>High Estimate</th>
<th>Low Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>Web Server</td>
<td>65%</td>
<td>55%</td>
</tr>
<tr>
<td>Sendmail</td>
<td>Email Client</td>
<td>76%</td>
<td>42%</td>
</tr>
<tr>
<td>Linux Server</td>
<td>Server Operating System</td>
<td>35%</td>
<td>28%</td>
</tr>
<tr>
<td>MySQL</td>
<td>Database</td>
<td>33%</td>
<td>29%</td>
</tr>
<tr>
<td>Firefox</td>
<td>Internet Browser</td>
<td>27%</td>
<td>10%</td>
</tr>
<tr>
<td>Linux Workstation</td>
<td>Client Operating System</td>
<td>3%</td>
<td>1%</td>
</tr>
<tr>
<td>OpenOffice</td>
<td>Office Productivity Suite</td>
<td>4%</td>
<td>2%</td>
</tr>
</tbody>
</table>

Source: Netcraft.com (Apache, Linux), Wikipedia.com (OpenOffice), Firefox.com, MySQL.com.
other internal employees less familiar with such software. Fourth, user organizations can consider outsourcing the implementation and support of open source applications to external vendors if they lack the manpower or the capability to do so themselves. This strategy can provide a quick way of overcoming the knowledge barriers associated with open source software adoption. Finally, organizations should consider sending key users to professional conferences and subscribing to trade magazines to keep current with the latest trends and events in this area.

Legacy Integration

Most organizations rely on legacy systems for mission-critical business processes such as accounting, sales, and manufacturing. These systems were created many decades ago using out-of-date technologies that do not interact well with today's technologies, are expensive to modify, and are irreplaceable given their key operational roles in organizations. Naturally, organizations are resistant to discarding or rebuilding their legacy systems and prefer new technologies that can integrate well and share data with existing legacy systems. Though legacy system integration is problematic for both open source and proprietary software, the prospect of inadequate legacy integration and lack of accountability for failed integration are often key reasons for organizations to avoid open source software adoption.

Though legacy integration has always been a thorny issue, new software-based mechanisms and standards are emerging to bridge the gap between existing legacy systems and new open source applications. Such integration can be accomplished using eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP) and related Web services standards, and Service-Oriented Architecture (SOA). Given that open source applications make their source code freely available for building extensions and modifications, the availability of integration tools from third-party vendors is much more likely for open source applications than for proprietary (closed-source) applications. Open source consultants and support services can assist with the selection of the right tools as well as customizing internal legacy systems to integrate well with contemporary open source applications.

Forking

Because open source software is developed by independent developers or groups of developers, there is always a possibility that each person or group may create their own version of software. Starting with the same source code, if different groups do not coordinate their efforts, the new features and functionality they add may not be interoperable with each other or exhibit equivalent functionality. “Forking,” as such code variations are called, was responsible for the fragmentation of the open source BSD-Unix community in the early 1990s into FreeBSD, OpenBSD, and NetBSD camps, allowing proprietary Microsoft Windows to establish a dominance in the operating systems space. Similar examples can be observed for the Emacs text editor, which forked in 1992 into GNU/emacs and Lucid/xemacs, and the NCSA Web server, which forked in 1995 into NCSA httpd (discontinued since) and Apache versions. Forking is inherently dangerous because it tends to fragment the adopter and applications market, hurting each version’s ability to gain a critical mass of adopters and forcing vendors to choose between the forked versions or create and support multiple versions of their own applications. Further, many potential adopters and vendors may take a “wait-and-see” approach for a dominant standard to emerge, before buying into the open source software, thereby stalling its adoption.

Some contend that forking is a natural evolution of the open source philosophy and may even have hidden benefits. The openness of the source code encourages developers to extend the core kernel to suit their own needs, and in doing so, may add to the functionality and long-term viability of the program. But the nature of open source licenses, such as the GNU General Public License (GPL), is such that if developers want to make these extensions “official” (for example, be part of the official kernel), they will have to transfer ownership of the source code to the open source community. The advantage of this approach is that if the original developer shuts down operations or otherwise decides to abandon a forked software and move on to more promising software, access to the source code will allow other developers to step in, maintain, and build extensions to the “orphaned” code and serve customers already using the forked code. This is not possible for proprietary software, where copyright restrictions will prevent others from maintaining or developing the orphaned software. This was the case of the GLUT library used in computer graphics, whose development stopped several years ago and no further development was possible to support advances in hardware and methodologies because the original software is still “protected by copyright.”

Fortunately, leading minds in the open source movement are aware of the dangers of forking and the importance of maintaining a common set of standards that developers can use to build and extend open source code. The recent merger of Open Source Development Labs (OSDL) and the Free Standards Group (FSG) to form The Linux Foundation and the formation of Open Source Initiative (OSI) standards body are right steps in this direction, which augers well for the future of open source software. Similar standards bodies, such as American National Standards Institute (ANSI) and the Institute of Electrical and Electronics Engineers (IEEE), have been central to providing leadership to and guiding the evolution of other computing technologies such as the structured query language (SQL) interface for relational databases (ANSI SQL-92) and wireless Ethernet protocols (IEEE 802.11g). In sum, though forking does present a hindrance for open source software adoption, adopting a forked software will not necessarily endanger business operations even if the original developer abandons code support or development (unlike proprietary software) and further, this problem appears to be self-resolving with the evolution of self-managed standards bodies.

Sunk Costs

Open source software is a fairly recent phenomenon starting in the 1990s, though organizations have been using software and applications since the 1960s. Many organizations have already
invested heavily in proprietary software prior to the emergence of open source, and adopting open source systems will require writing off prior investments as a “sunk cost.” Naturally, many organizations are unwilling to bear this cost, and hence cannot adopt open source systems on an enterprise wide scale. Since organizational executives demand cost justification for most new technology investments, the sunk cost of existing proprietary systems renders open source adoption unjustifiable.

Organizations considering open source adoption may attempt to address the sunk cost problem in two ways. First, they may consider partial adoption of open source software, in specific areas where no prior proprietary system (and hence, sunk cost) exists. For instance, an organization without business intelligence applications may consider open source variants of these applications such as Jasper Reports and Pentaho (in lieu of proprietary variants), even though it may wish to retain proprietary versions of sales, accounting, manufacturing, and other mission-critical business systems already in place. Second, instead of comparing the adoption costs of open source with the sunk costs of proprietary applications, organizations should compare the future cost streams of maintaining their proprietary applications vis-à-vis open source applications. The cost savings in licensing and using open source systems over a multiple-year period may potentially be adequate to justify the sunk costs invested in prior proprietary systems.

Perception of Technological Immaturity
There is a common perception among many organizational managers that open source software is an immature technology and not yet ready for commercial use. Many also believe that goods available for free, such as open source software, are probably of inferior quality than those that are paid for, such as proprietary software. Fueling this perception may be the “bazaar style” development of open source software, by volunteer developers at their leisure and without formal oversight or incentives, and the fear that one day, these developers may not want to continue improving these systems for free or may move on to other technologies. Not surprisingly, such thoughts are frequently echoed by the marketing divisions of proprietary software companies to plant uncertainty and doubt in the minds of potential open source software adopters, in an effort to retain competitive advantage for their own products.

However, the hundreds of thousands of downloads to date of popular open source software such as Linux, Apache, and MySQL from Sourceforge, net should put to rest fears about their technological maturity or their long-term sustainability. Further, the fact that commercial organizations such as IBM, Hewlett-Packard, and Sun Microsystems are investing millions of dollars on open source software development should provide some degree of confidence in the minds of concerned adopters. Further, open source software is fast gaining in maturity, possibly more so than their proprietary counterparts. Third-party organizations have developed formal evaluation metrics and models to benchmark the quality and robustness of open source software. These models include open source maturity models by Navica, CapGemini, and Atos Origin. These models consider a wide range of factors such as the availability of training, documentation, third party support, integrated software and other professional services, community size, community age, and lines of source code, with different weights for each factor, to estimate the maturity of open source software. Potential adopters considering open source adoption may employ the above models to evaluate for themselves the true quality and maturity levels of their targeted software.

Conclusion
While the benefits of open source software, such as cost savings, vendor independence, and open standards, are often highlighted in the practitioner press, the barriers confronting open source software adoption and potential ways of overcoming these barriers are less known. This article described five major barriers for adopting such software, along with potential remedies for each barrier. We hope that our analysis will help user organizations make a fair and balanced assessment of the benefits and challenges of open source adoption, when they are confronted with that decision.

References
When we call software “free,” we mean it respects the users' essential freedoms: the freedom to run it, to study and change it, and to redistribute copies with or without changes (see http://www.gnu.org/philosophy/freesw.html). This is a matter of freedom, not price, so think of “free speech,” not “free beer.”

These freedoms are vitally important. They are essential, not just for the individual users' sake, but because they promote social solidarity—that is, sharing and cooperation. They become even more important as more aspects of our culture and life activities are digitized. In a world of digital sounds, images, and words, free software increasingly equates with freedom in general.

Tens of millions of people around the world now use free software; the schools in regions of India and Spain now teach all students to use the free GNU/Linux operating system (see http://www.gnu.org/gnu/linux-and-gnu.html). But most of these users have never heard of the ethical reasons for which we developed this system and built the free software community, because today this system and community are more often described as “open source,” and attributed to a different philosophy in which these freedoms are hardly mentioned.

The free software movement has campaigned for computer users' freedom since 1983. In 1984 we launched the development of the free operating system GNU, so we could avoid the non-free operating systems that deny freedom to their users. During the 1980s, we developed most of the essential components of such a system, as well as the GNU General Public License (see http://www.gnu.org/licenses/gpl.html), a license designed specifically to protect freedom for all users of a program.

However, not all of the users and developers of free software agreed with the goals of the free software movement. In 1998, a part of the free software community splintered off and began campaigning in the name of “open source.” The term was originally proposed to avoid a possible misunderstanding of the term “free software,” but it soon became associated with philosophical views quite different from those of the free software movement.

Some of the proponents of “open source” considered it a marketing campaign for free software, which would appeal to business executives by citing practical benefits, while avoiding ideas of right and wrong they might not like to hear. Other proponents flatly rejected the free software movement's ethical and social values. Whichever their views, when campaigning for “open source” they did not cite or advocate those values. The term “open source” quickly became associated with the practice of citing only practical values, such as making powerful, reliable software. Most of the supporters of “open source” have come to it since then, and that practice is what they take it to mean.

Nearly all open source software is free software; the two terms describe almost the same category of software. But they stand for views based on fundamentally different values. Open source is a development methodology; free software is a social movement. For the free software movement, free software is an ethical imperative, because only free software respects the users' freedom. By contrast, the philosophy of open source considers issues in terms of how to make software “better”—in a practical sense only. It says that non-free software is a suboptimal solution. For the free software movement, however, non-free software is a social problem, and moving to free software is the solution.

Free software. Open source. If it's the same software, does it matter which name you use? Yes, because different words convey different ideas.
While a free program by any other name would give you the same freedom today, establishing freedom in a lasting way depends above all on teaching people to value freedom. If you want to help do this, it is essential to speak about “free software.”

We in the free software movement don’t think of the open source camp as an enemy; the enemy is proprietary (non-free) software. But we want people to know we stand for freedom, so we do not accept being misidentified as open source supporters.

Common Misunderstandings of “Free Software” and “Open Source”

The term “free software” has a problem of misinterpretation: an unintended meaning, “software you can get for zero price,” fits the term just as well as the intended meaning, “software that gives the user certain freedoms.” We address this problem by publishing the definition of free software, and by saying “Think of free speech, not free beer.” This is not a perfect solution; it cannot completely eliminate the problem. An unambiguous, correct term would be better, if it didn’t have other problems.

Unfortunately, all the alternatives in English have problems of their own. We’ve looked at many alternatives that people have suggested, but none is so clearly correct that switching to it would be a good idea. Every proposed replacement for “free software” has some kind of semantic problem—and this includes “open source software.”

The official definition of “open source software,” which is published by the Open Source Initiative (see http://opensource.org/docs/osd) and too long to cite here, was derived indirectly from the Open Source Initiative (see http://opensource.org/docs/osd) and too long to cite here, was derived indirectly from the state of Kansas published a similar project to develop a free replacement.”

Common misunderstandings of the term “open source” are that it means “You can look at the source code,” and that most people seem to think that’s what it means. That is a much weaker criterion than free software, and much weaker than the official definition of open source. It includes many programs that are neither free nor open source. Since that obvious meaning for “open source” is not the meaning that its advocates intend, the result is that most people misunderstand the term. Here is how writer Neal Stephenson defined “open source”: Linux is “open source” software meaning, simply, that anyone can get copies of its source code files.

I don’t think Stephenson deliberately sought to reject or dispute the “official” definition. I think he simply applied the conventions of the English language to come up with a meaning for the term. The state of Kansas published a similar definition: Make use of open-source software (OSS). OSS is software for which the source code is freely and publicly available, though the specific licensing agreements vary as to what one is allowed to do with that code.

Open source supporters try to deal with this by pointing to their official definition, but that corrective approach is less effective for them than it is for us. The term “free software” has two natural meanings, one of which is the intended meaning, so a person who has grasped the idea of “free speech, not free beer” will not get it wrong again. But “open source” has only one natural meaning, which is different from the meaning its supporters intend. So there is no succinct way to explain and justify the official definition of “open source.” That makes for worse confusion.

Another common misunderstanding of “open source” is the idea that it means “not using the GNU GPL.” It tends to accompany a misunderstanding of “free software,” equating it to “GPL-covered software.” These are equally mistaken, since the GNU GPL is considered an open source license, and most of the open source licenses are considered free software licenses.

Different Values Can Lead to Similar Conclusions… But Not Always

Radical groups in the 1960s had a reputation for factionalism: some organizations split because of disagreements on details of strategy, and the two resultant groups treated each other as enemies despite having similar basic goals and values. The right wing made much of this, and used it to criticize the left entirely.

Some try to disparage the free software movement by comparing our disagreement with open source to the disagreements of those radical groups. They have it backward. We disagree with the open source camp on the basics of goals and values, but their views and ours lead in many cases to the same practical behavior—such as developing free software.

As a result, people from the free software movement and the open source camp often work together on practical projects such as software development. It is remarkable that such different philosophical views can so often motivate different people to participate in the same projects. Nonetheless, these views are very different, and there are situations where they lead to very different actions.

The idea of open source is that allowing users to change and redistribute the software will make it more powerful and reliable. But this is not guaranteed. Developers of proprietary software are not necessarily incompetent. Sometimes they produce a program that is powerful and reliable, even though it does not respect the users’ freedom. How will free software activists and open source enthusiasts react to that?

A pure open source enthusiast, one that is not at all influenced by the ideals of free software, will say, “I am surprised you were able to make the program work so well without using our development model, but you did. How can I get a copy?” This attitude will reward schemes that take away our freedom, leading to its loss.

The free software activist will say, “Your program is very attractive, but not at the price of my freedom. So I have to do without it. Instead I will support a project to develop a free replacement.”
If we value our freedom, we can act to maintain and defend it.

**Powerful, Reliable Software Can Be Bad**

The idea that we want software to be powerful and reliable comes from the supposition that the software is designed to serve its users. If it is powerful and reliable, that means it serves them better.

But software can only be said to serve its users if it respects their freedom. What if the software is designed to put chains on its users? Then powerlessness only means the chains are more constricting, and reliability that they are harder to remove. Malicious features, such as spying on the users, restricting the users, back doors, and imposed upgrades are common in proprietary software, and some open source supporters want to do likewise.

Under the pressure of the movie and record companies, software for individuals to use is increasingly designed specifically to restrict them. This malicious feature is known as DRM, or Digital Restrictions Management (see DefectiveByDesign.org), and it is the antithesis in spirit of the freedom that free software aims to provide. And not just in spirit: since the goal of DRM is to trample your freedom, DRM developers try to make it difficult, impossible, or even illegal for you to change the software that implements the DRM.

Yet some open source supporters have proposed “open source DRM” software. Their idea is that by publishing the source code of programs designed to restrict your access to encrypted media, and allowing others to change it, they will produce more powerful and reliable software for restricting users like you. Then it will be delivered to you in devices that do not allow you to change it.

This software might be “open source,” and use the open source development model; but it won’t be free software, since it won’t respect the freedom of the users that actually run it. If the open source development model succeeds in making this software more powerful and reliable for restricting you, that will make it even worse.

**Fear of Freedom**

The main initial motivation for the term “open source” is that the ethical ideas of “free software” make some people uneasy. That’s true: talking about freedom, about ethical issues, about responsibilities as well as convenience, is asking people to think about things they might prefer to ignore, such as whether their conduct is ethical. This can trigger discomfort, and some people may simply close their minds to it. It does not follow that we ought to stop talking about these things.

However, that is what the leaders of “open source” decided to do. They figured that by keeping quiet about ethics and freedom, and talking only about the immediate practical benefits of certain free software, they might be able to “sell” the software more effectively to certain users, especially businesses.

This approach has proved effective, in its own terms. The rhetoric of open source has convinced many businesses and individuals to use, and even develop, free software, which has extended our community—but only at the superficial, practical level. The philosophy of open source, with its purely practical values, impedes understanding of the deeper ideas of free software; it brings many people into our community, but does not teach them to defend it. That is good, as far as it goes, but it is not enough to make freedom secure. Attracting users to free software takes them just part of the way to becoming defenders of their own freedom.

Sooner or later these users will be invited to switch back to proprietary software for some practical advantage. Countless companies seek to offer such temptation, some even offering copies gratis. Why would users decline? Only if they have learned to value the freedom free software gives them, to value freedom as such rather than the technical and practical convenience of specific free software. To spread this idea, we have to talk about freedom. A certain amount of the “keep quiet” approach to business can be useful for the community, but it is dangerous if it becomes so common that the love of freedom comes to seem like an eccentricity.

That dangerous situation is exactly what we have. Most people involved with free software say little about freedom—usually because they seek to be more acceptable to business. Software distributors especially show this pattern. Nearly all GNU/Linux operating system distributions add proprietary packages to the basic free system, and they invite users to consider this an advantage, rather than a step backward from freedom.

Proprietary add-on software and partially non-free GNU/Linux distributions find fertile ground because most of our community does not insist on freedom with its software. This is no coincidence. Most GNU/Linux users were introduced to the system by “open source” discussion, which doesn’t say freedom is a goal. The practices that don’t uphold freedom and the words that don’t talk about freedom go hand in hand, each promoting the other. To overcome this tendency, we need more, not less, talk about freedom.

**Conclusion**

As the advocates of open source draw new users into our community, we free software activists must work even more to bring the issue of freedom to those new users’ attention. We have to say, “It’s free software and it gives you freedom!”—more and louder than ever. Every time you say “free software” rather than “open source,” you help our campaign.

**Further Reading**

1. Joe Barr wrote an article called “Live and Let License” (see http://www.itworld.com/LW031102/control4) that gives his perspective on this issue.
2. Lakhani and Wolf’s paper on the motivation of free software developers (see http://freesoftware.mit.edu/papers/lakhaniwolf.pdf) states that a considerable fraction are motivated by the view that software should be free. This was despite the fact they surveyed the developers on SourceForge, a site that does not support the view that this is an ethical issue.

Richard Stallman (rms@gnu.org) is the author of the free symbolic debugger GDB, the founder the project to develop the free GNU operating system, and the founder of the Free Software Foundation.

Copyright © 2009 Richard Stallman

Verbatim copying and distribution of this entire article is permitted in any medium, provided this notice is preserved.

**Software can only be said to serve its users if it respects their freedom.**