4
Open Source Baselines: Compared to What?

Lawrence Lessig

Software is the set of instructions that makes a computer run. Programmers or coders write these instructions. The instructions then get translated into a form that computers can understand. The product of the initial authoring is called source code; the product of the translation is called object code. Humans write source code; machines run object code. Well-trained humans can read and understand source code; superhumans and computers read and process object code.

When software is distributed, the distributor makes a choice about whether the distribution will include both the source and object code. "Proprietary software" refers to distributions of software that include just the object, or binary, code. "Open source and free software" refers to distributions of software that include both the object and source code. With proprietary software, the consumer receives a program that he or she can run. With open source or free software, the consumer receives a program that he can run, modify, and—depending upon the license under which the program is received—redistribute. Proprietary software is like Kentucky Fried Chicken. Open source and free software is like Kentucky Fried Chicken sold with the "original secret recipe" printed in bold on the box.

And thus here lies the puzzle: by distributing the source code with the object code, open source and free software developers give their competitors free access to any value that they might have added to the software they are distributing. A developer thus cannot capture that value for him- or herself, but rather gives at least a part of it away. How then can developers have sufficient incentive to innovate? What motivates them to develop in this way? How can developers sustain the costs of development if they must hand to their competitors all the value they have created?

My aim here is to disentangle this puzzle. Open source and free software have played an important part in the growth of the Internet. The puzzle about their existence comes from a mistaken baseline of comparison. Properly understood, these movements are completely consistent with a tradition of innovation and development outside the context of software. They may seem unique within the software industry, but they are not unique against the background of development or innovation generally.

Using this appropriate baseline for comparison, the following discussion develops an account of the social value of open source and free software. This account, in turn, supports an argument in favor of the government adopting and supporting open source and free software projects.

Open Source and Free Software Defined

To the extent that there is sustained opposition to open source and free software, that opposition is targeted and quite narrow. Microsoft has been the most vocal opponent of a particular flavor of open source and free software development, an opposition that is subtle and, properly understood, specific.[1] But understanding this opposition and its relatively narrow scope requires a bit of background about the nature of free and open source software. And understanding the countervailing benefits from free and open source software in turn requires a bit of background about the relationship between software and what economists call "public goods."

The Nature of Open Source and Free Software. Open source and free software give consumers and the public something more than proprietary software does: the ability to tinker and modify. Such software gives the public the benefit of the information contained within the code. Yet open source and free software don't provide these values by forfeiting public law protection. Open source and free software are not "in the public domain." Copyrights still attach to their creative content. Thus copyright law continues to control how this content can be used and distributed. Open source and free software producers use this control to impose conditions upon the use of their code. These conditions vary significantly depending upon whether the code is free or open source. But these conditions are not options. They are requirements imposed by the force of law.

Not all software-related content is protected in this way. There are important software related products that are within the public domain. The TCP/IP (Transmission Control Protocol/Internet Protocol), for example, which forms the basic protocols of the Internet, is in the public domain. Anyone is free to implement it without the permission of a copyright holder. This enabled many to build TCP/IP networks inexpensively and ensured that no one had the power to control how TCP/IP would develop.

But being in the public domain also means that TCP/IP could in principle be hijacked. A major producer of TCP/IP technology could extend the protocol in a way that benefits its own interests and weakens its competitors. It could do this because the nature of the public domain is that anyone is free to build as they wish. HTML (Hypertext Markup Language) is an example of a protocol that was in the public domain. Netscape and Microsoft each tried to extend the protocol in ways that benefited its own implementation.[2] This competition may or may not have been beneficial to the spread of the World Wide Web. But whether or not it was, hijacking was possible because the underlying protocol was not protected.

By staying outside the public domain, open source and free software at least have the potential to protect themselves against the hijacker. Using copyright law, they have the power to require certain conditions before their code is used in ways that implicate the exclusive rights protected by copyright law. Thus, like proprietary software, open source and free software depend upon copyright; like proprietary software, open source and free software make themselves available only under certain conditions. The important difference among these three forms of software is simply the difference in conditions.

Proprietary software is made available upon the payment of a price (which sometimes is zero). In exchange for a price, the user ordinarily licenses the object code. Object code, because it is compiled into a form that is effectively opaque to humans, does not transmit the information it contains; it is simply a machine that induces another machine to function in a particular way. But attached to that machine is a license supported by copyright law. That license sets the terms according to which one may use the licensed machine. In the ordinary proprietary model, you are not permitted to sell the code you have licensed, nor are you permitted to modify and redistribute it. Proprietary code gives you the right to use the machine you've licensed, just like a rental from Hertz gives you the right to use the car you've leased.

Open source and free software impose different conditions upon users. And while the variety of open source and free software licenses is broad, we can identify essentially two sorts: copylefted software and noncopylefted software.[3]

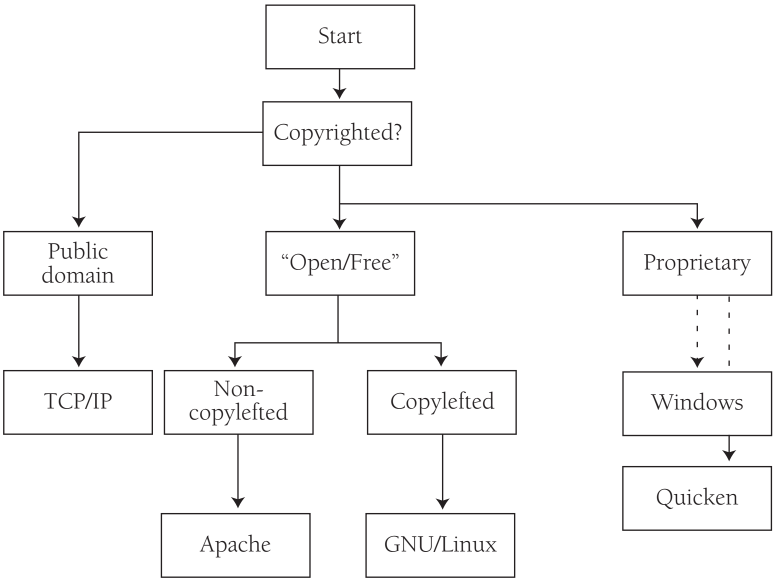
Copylefted software is software that is licensed under terms that require follow-on users to require others to adopt the same license terms for work derived from the copylefted code. The principle is "share and share alike." Noncopylefted open source software imposes no such condition on subsequent use. With copylefted software, the price of admission is that if you redistribute modified versions of the copylefted code, you must redistribute it under similar license terms; with noncopylefted software, no such price is demanded.

The most famous example of copylefted code is the GNU/ Linux operating system.[4] GNU/Linux is licensed under the GNU General Public License (GPL). The GPL requires that anyone who modifies and redistributes GPL-covered code do so under a GPL license.[5] For example, if an enterprising coder modified Linux to run seamlessly Windows and Macintosh programs, he would be free to redistribute that modified GNU/Linux only if he did so under a GPL license. And since a GPL license also requires that the source code of a GPL work be made available for free, this Linux innovator would likely face competition from copycat competitors. If the coder had in fact produced an operating system that could run programs from other operating systems directly, then many would likely take it and sell it in competition with him. The GPL guarantees that "freedom."

It is for this reason that some argue that the copyleft requirement is too steep a price for developers to pay.[6] The freedom of a single developer to build a cross-platform-compatible version of Linux, for example, might well be defeated by the copyleft condition (assuming that the costs of such a project are extremely high and that the developer would need to recover those costs from the sale of copies of the resulting operating system). But this condition is not necessarily any more expensive than the conditions imposed by proprietary code. If our Linux developer wanted to create a modified, cross-platform-compatible version of Windows, he would be no more free to redistribute the resulting Microsoft code than he would be free to distribute the modified Linux code under the GPL. If he could get the permission of Microsoft at all, no doubt he would have to pay a high price. The difference then is not that one licensing system imposes burdens while the other does not; the difference is in the nature of the burdens.

Noncopylefted open source software does not impose this condition on subsequent licensing. Not only is a user free to build upon it, but it also does not require that such building be released under similar licensing terms. The Apache web server is an example of this kind of software. Apache is an open source

Figure 4-1. Categories of Software



server, the most widely used web server in the world.[7] Anyone is free to download and modify the source code for the Apache server. And because Apache is not copylefted, anyone can take the source code and build it into his own software project. There is nothing in the Apache license that would prohibit a company from taking the Apache code, compiling it, and then selling the resulting server as a proprietary product (so long, at least, as it was not called "Apache"). These distinctions are mapped in figure 4-1.

No doubt, from a private perspective, the differences among these types of software are important. As you move from left to right on the chart, the ability of coders to capture the value of what they code increases. It doesn't follow, of course, that the income coders receive necessarily increases the further to the right they place their code, for the terms of the license could well affect the value of the software to others. One percent of a million is greater than 100 percent of a thousand. But as you move to the right, the legal right of coders to extract the full value from their code certainly does improve.

From the perspective of the public, however, another difference is significant. To understand this difference, however, requires a bit more explanation about the economics of software.

Software and Public Goods. As discussed above, software is a set of organized instructions for making a computer function. These instructions are written first in source code, which is then compiled into object code. The source code is understandable by the humans who write it; the object code is understandable by computers.[8] The product of these writings—a digital object stored on a hardware device—can then be copied at almost no cost. With the emergence of powerful networks, it can also be distributed at almost no cost.

This means that to some degree software has the attributes of "public goods."[9] A public good is not just any good that benefits the public. Food is a good that benefits the public; it is not a public good. As economists define the term, a public good is a resource that is both nonrivalrous and nonexcludable. It is nonrivalrous if your consumption of the resource does not reduce the amount available to me. It is nonexcludable if there is no feasible way to block you from consuming the resource once it is made available to me. National defense is the classic public good. Whatever level of spending the government provides for national defense, my consumption of national defense does not reduce the amount available to you. And if the nation is properly defended, I get that benefit whether or not I have contributed to its supply.[10]

Goods can be mixed—possessing both public and private attributes. Software is an example: it can be produced as a purely private good, but it can also be produced in a way that promotes certain public goods. These public goods might be divided between pure and "qualified" public goods. The information about how a program works—how it achieves its functionality—that is contained within its source code is a pure public good. If made available generally, then my consumption of that knowledge would leave as much for you as before. If made available generally, then it would be hard to exclude my knowledge of it to the extent that knowledge is known generally.

In contrast, the digital copy of a particular software product could be considered a qualified public good, meaning simply that it requires some resources in order to be obtained. My having a copy of your program doesn't interfere with your having a copy of your program. But it may take resources to produce that copy of your program. Likewise, an unprotected digital copy can be made available to all if it is made available to some, but it takes resources to move that content (the cost of electricity to run the network, for example). As modern digital technologies reduce these necessary resources to zero, this means in effect that software can be made available just as easily as a pure public good.[11]

The economics of public goods is, of course, well developed. If, following Elinor Ostrom, we distinguish between the *provision* of a public good and its *consumption*, then neoclassical economics tells us that social welfare is maximized if an *already existing* public good gets consumed at a price equal to its marginal cost of production.[12] That doesn't mean an economic system should require that its price be its marginal cost. If public goods could earn no more than their marginal cost of production, there would be an insufficient incentive to produce at least some public goods. If all the good that Microsoft produced by producing Windows XP could be immediately expropriated by every Microsoft wannabe at the marginal cost of producing a copy of Windows XP, then there'd be little incentive for Microsoft to build many of its products. If every song that Britney Spears recorded could be distributed to others for free, there'd be little incentive for Britney Spears to record as she does. With these public goods at least, solving the consumption problem (by making the goods available at zero marginal cost) creates a provision problem (by not creating enough return to support the incentive to create). Thus to maintain incentives to produce, such public goods require a way to defeat at least part of their own nature as public goods. In other words, what is needed is a way to make at least these public goods rivalrous or excludable, so that a price above the marginal cost can be collected.[13]

The law has long provided a device to achieve precisely this end—intellectual property. Intellectual property gives authors the right to control the distribution of copies of their work. That right balances the public goods character that writing makes manifest. The law levels the playing field for the publisher by giving authors the exclusive right to copies of their original work. Then de jure the work becomes rivalrous and excludable.

Software developers, however, have tools beyond intellectual property law that they can deploy to balance the public goods nature of software. Proprietary software providers, for example, can add excludability to their software by never releasing its source code: by compiling the source code and distributing just its object code, they can make the information within the product effectively excludable. Compilation makes the source code secret, and secrecy adds to the providers' ability to recover value.

Software providers can also make software effectively rivalrous. Copy protection technologies, for example, can make particular copies of software exclusive to particular owners. Properly deployed, these technologies can make it effectively impossible for you to use my software when I am using it. This again makes it easier for the producer to recover the value of his production by eliminating another public goods aspect of the digital product.

Thus two sets of tools—one public, the other private—are available to the software provider for balancing (or defeating) the public goods character of code. These tools in turn overlap. An author doesn't have to choose between copy protection technologies and copyright or between compiled code and law—he gets both.[14] And by combining both kinds, a producer will get more value and will, to some extent and in some contexts, increase the incentive to produce.

Yet at some point, the combination of this public and private protection may reduce, rather than increase, social welfare—at least if the protection is too strong. These protections raise the price of information above its marginal cost of production. This means less information is being distributed than is economically efficient. Economics resents a price above marginal cost. Any gap may be a necessary evil to induce production, but as with all necessary evils, it should be tolerated only so far as it is truly necessary. A gap may be justified by the need to solve the provision problem, but if the controls extend beyond the justification, they reduce social welfare. Some control is needed, but some control is far less than perfect control. And hence the problem of social policy is how best to balance a necessary evil against access to information at its marginal cost—that is, free.[15]

These familiar ideas are presented here to remind us of a point that is too often forgotten in the debate about open source and free software: the strong bias of public policy should be to spread public goods at their marginal cost. Compromises are no doubt necessary if private actors are to contribute voluntarily to the production of public goods; but public entities, such as governments, should not indulge in these compromises unless they are necessary. Between two systems for producing a public good, one that releases the information produced by that good freely and one that does not, all things being equal, public policy should favor free access. This is not because of some egalitarian bias or because of ideals about social equality but for purely neoclassical economic reasons: free access brings the cost of information down to its marginal cost, and neoclassical economics favors price at marginal costs. If the problems of incentives have already been solved for a particular good or class of goods—no doubt a large assumption, but for some important software goods a true one—then there is no further reason to exclude access to the public goods produced.[16] Or if the provision problem is sufficiently solved by other systems of incentive, then again there is no reason to exclude the public goods produced.

From a social perspective, this means that there is a difficult choice between these two forms of production. If social good is the sum of private and public goods, then we cannot pick between open and proprietary software in the abstract. On the one hand, open source and free software dominate proprietary software in spreading a public good. Yet on the other hand, if there is insufficient incentive to produce software under the open source and free software models, then the private good from software will also be underproduced.

When society faces a difficult social choice, that is usually a good reason to let both forms of production compete in the market. And thus no one sensible is calling for a requirement that all software be free or that free software be banned. Yet some do argue against open source and free software, sometimes motivated by a belief that its business model is a failure and sometimes motivated by a view that at least some forms of free software are dangerous to "software ecology."

The economics of open source and free software is just beginning to be understood. A growing body of literature increasingly demonstrates how individuals could have sufficient private incentive to solve the provision problem, even though they cannot capture the full value of what they produce.[17] The aim here is not to comment upon that work within economics proper but rather to clear the way for the lessons that economics is increasingly offering in the realm of public policy.

The following section connects the practice of open source and free software development to other more familiar instances of production. That link in turn will make open source development more familiar. It may well be that proprietary software maximizes the private return from software creation (though even that is questioned by some). But it does not follow that proprietary production is the only form of production that provides sufficient incentives for individuals and corporations to contribute to its supply. In this case, as in many others, an imperfect ability to capture the value of innovation does not necessarily destroy the incentive to innovate. And more important, increasing the ability to capture the value of innovation does not necessarily increase the incentive to innovate.[18]

Parallels

The system of production that produces open source and free software is not an exception within a free society. Properly understood, it stands on a continuum with production in most competitive fields. It is instead proprietary software that is outside the norm: its system of production is the exception, not the rule.

This positive claim does not entail negative implications for proprietary code. Its intent is rather to challenge a common assumption regarding open source or free software: that it is an exception within a free market, more akin to communism than capitalism. This view is mistaken.

Open source and free software are systems of production that mix private resources with those in a commons to produce goods or services of economic value. The production function thus includes both private and public goods, where the labor used to produce new code is a private good, and the information used to build upon the free software is public. The innovation and utility that this software produces benefits both the creator and the public. The producer captures some—but not all—of the value he or she produces.

Viewed in this way, it is easy to see that most production in a free market is structured in precisely the same way. Think about a coffee shop that opens across the street from Starbucks. It gets to draw upon lots of resources held in common—knowledge about how to attract customers, a taste for coffee, a taste for high-priced coffee—even though many of these resources might be directly attributable to Starbucks. Though Peet's Coffee claims to be older, it was certainly Starbucks that created the American norm for high-priced coffee.[19] That norm is certainly of value to Starbucks' competitors, but we don't therefore automatically assume that Starbucks has a claim to the value produced by this norm. There's no general principle within a free market that says that every quantum of value produced is the property of the entity that produced it. The value that Starbucks gets to reap from its innovation in coffee sales is just the amount it can make in the free market. As economists have long taught, the amount a business such as Starbucks makes in the market is a function of the demand for its product, the marginal cost of production, and competition. These may or may not equal the total value that Starbucks has contributed, but neoclassical economics was born with the insight that value received is not necessarily equivalent to the value produced.

Or consider a second example that is more familiar to the lawyer.[20] Think about the system of production called legal litigation services. Lawyers involved in litigation write texts to be submitted to courts; courts in response produce other texts that are published and made available to the greater society. The texts within this system are creative works (some more creative than others), but they are all fully available for others to draw upon and copy.[21] Opinions of judges can be used without the permission of the court; arguments from appellate briefs are fair game for others to use later on. All these resources are "free resources" in just the sense that free and open source software produces free resources. But none would conclude from this that lawyers are underpaid or that the legal system does not provide adequate incentives to lawyers. Instead the task of the lawyer is to mix new work with these existing resources to produce a result that benefits the client. The lawyer is paid for this mixing, even though the value of the mix is available for others to take.

The vast majority of coding in software projects has precisely the same character. As James Bessen has argued, most coding is customization—fitting existing resources to new uses.[22] Those who customize generate a product that is tailored to a particular user. Ordinary contract law gives the customizer adequate power to ensure a return to support that customization. Software coders in general are better off if there is a great deal of free software for them to draw upon. As with lawyers, there is no good that comes from forcing everyone to reinvent the wheel. And thus there is a public benefit if customizers contribute their work into a commons while also being paid for their particular customization.

These anecdotes point to a more general truth that, as William Baumol has quantified it, the vast majority of the value of innovation in a free market is not captured by the innovator. In other words, most of the value "spills over" into the market generally. And while sometimes reducing spillover will increase the efficiency of the market, as Baumol has argued, this assumption is not always correct. Sometimes the spillover is efficient.[23]

None of this is meant to argue that every software project should be open or free. Nor does it follow that lawyers should be unable to protect particular forms of their craft. My point is to show the ordinariness of systems of production where the producer does not fully capture the value he produces. That system is the norm in a free society. We call it a free market. It is not "theft" each time it is realized, and when it is realized, it can induce public benefit.

An example of this social benefit is seen today in the context of embedded systems.[24] As technologists find ways to build computers into every device around us, they need an operating system to run those computers. The free operating system of GNU/Linux is the overwhelmingly dominant operating system deployed in these embedded systems. This is for obvious reasons. An embedded system producer wants to minimize costs, which include both the acquisition costs of software and the maintenance or debugging costs of that software. Because the GNU/Linux operating system is free, the acquisition costs are low, and because many people have had an opportunity to debug the system, its reliability is extremely high. Thus the GNU/Linux operating system is an inexpensive and valuable resource for the embedded systems market. Without this inexpensive resource, the embedded market would not have experienced such a high degree of growth.

Others have suggested that there are more systematic benefits accruing to open source or free software projects.[25] As James

Bessen has argued, the costs of debugging complex projects such as software are so high that an open source project will often be able to bear those costs better than a proprietary project.[26] No doubt in many cases this claim is correct, as are other claims about the private benefit that open source or free software projects enjoy relative to proprietary projects. These are reasons why open source or free software projects could have a competitive advantage over proprietary projects.

Whether they have an advantage or not, however, my point is more modest. The phenomenon of open source coding is just one instance of a more general and familiar mode of production. Like most innovation within a free society, innovators cannot capture all the value they produce. The only relevant difference with open source and free software is that it chooses this "imperfection" whereas the others may not have any choice.[27]

Lessons

Finally, there is the question of government's attitude toward open source or free software. There has been a trend in many countries for the government to insist that its agencies use open source or free software. France, Germany, and China have recently announced such policies.[28] This, in turn, has pushed many in the United States to argue for the opposite. David Evans, for example, argues against governments favoring open source or free software in their purchasing decisions.[29] Microsoft, too, has argued against developing nations adopting GNU/Linux.[30]

Evans maintains that the government should make decisions about whether to adopt open source or free software in the same way as private actors should: the only question it should ask is what software maximizes efficiency.[31] I agree with his principle, but we must be careful about how it is applied to the particulars of the case, because the factors that determine efficiency for governments are fundamentally different from the factors that determine efficiency for private actors. Governments are not competitors in the sense that private actors are. They have a greater interest in externalizing benefits that other competitors might share.

Consider, for example, a government that is funding the development of an email system for its employees. That others could use the same code within the government if it were open source or free software would be one reason for the government to use such software to develop its system. Its code will produce a benefit for other governmental actors. All things being equal, that benefit should weigh in favor of the open source over proprietary code.

The same is true at the platform level. If the choice of a platform produces a positive externality for others within the government, this is a reason, all things considered, for the government to choose the software producing that externality. This is, for example, the argument used most commonly in favor of the government purchasing Windows as its base operating system. Uniformity throughout the government has its benefits. But it would also be beneficial to have uniformity at a platform level that was open. My point is simply that such benefits should be accounted for by the government as well.

Microsoft has questioned the propriety of government funding development of software covered by the GPL. In a series of public statements, the company has opposed the government's "support" for "free software," meaning software licensed under the GPL.[32] As explained earlier, open source software is not licensed under the GPL. Thus, Microsoft's argument says nothing about whether the government should adopt, for example, Apache web servers. The only target of Microsoft's attack is the GPL.

Microsoft's arguments are understandable, though how far they extend is hard to know. Microsoft argues that software licensed under the GPL has a "viral" tendency, as modifications to that software must themselves be licensed under the GPL. This means that some companies (in particular, some proprietary producers of software) are unable to use GPL code. This, in turn, could mean that projects operating under a GPL would become unavailable to companies such as Microsoft. GPL-based projects would therefore be developed and worked on only by GPL-friendly enterprises, which, Microsoft argues, could reduce the incentive to develop software.

At a formal level, one might well make the same charge against the government funding the development of proprietary software.[33] After all, if the government funds a proprietary software project, only the company owning or licensing the copyrights can participate in further development of that software. Other companies without access to the proper permissions are banned from developing this software. So just like Microsoft's claims about GPL software, these companies are precluded from participating in proprietary-software-based projects. By a parity of reasoning, it would therefore follow that the government should not fund proprietary projects either. By this line of reasoning, the only software projects the government should fund would be those that produce code in the public domain or under licenses similar to the dominant open source licenses, such as Apache.

As an argument about fairness, this ultimate conclusion might well have some force. But as an argument of efficiency, it doesn't quite work. If the test the government applies is which software benefits the government most at a given price, then it is not clear why the fact that it would exclude some private companies from developing that software should, on its own, matter. It could under certain assumptions matter, but in the abstract, it is hard to see just why the licensing choices of potential competitors should constrain the government in its choice of code.

Properly calibrated, then, government neutrality could well entail a preference for open or free software, depending on the program and the interests involved. More specifically, between two products, open source and proprietary, of comparable strength, there is a reason for the government to prefer an open source version. And between two products, free and proprietary, of comparable strength, there may even be a reason for the government to prefer the free software. This conclusion would not always follow, but it would follow for a wide range of code that today is not open source or free software.

There is a second type of neutrality that governments should also consider. This touches the question of software patents.[34]

There is not the space here to consider the full range of arguments surrounding the software patent debate. In my view, the debate shows conclusively that while there are clear costs to innovation imposed by software patents, there is no good evidence that they provide a sufficiently strong countervailing benefit.

The costs of patents, however, are significantly greater for open source and free software projects than they are for proprietary projects. The transaction costs of licensing are higher with open source and free software; the ability to license a patent is therefore decreased. This means that a system with software patents is biased against open source and free software. That bias might be justified if there were any strong indication that software patents do any good. But with the evidence pointing the other way, this is a clear example of partiality in which the government should not engage. Models of software development should be allowed to compete; the government should not allow bloated intellectual property regimes to tilt the field of competition against one of the most vibrant competitors.

Conclusion

There is a reason for public policy to prefer a world where software is open and free. That reason cannot trump all other considerations, and it alone does not support a general rule that would banish proprietary code. But the reason does motivate an inquiry into whether free and open source code can be adequately produced. Economists have just begun a formal inquiry into that question. That inquiry could be helped with a bit of perspective: by seeing the parallel between open source and free software production and other more familiar modes of production, we are more likely to accept the conclusion of economists that open code is often possible and often very valuable.

I have argued in favor of government neutrality regarding open and proprietary software—as long as the interests the government reckons are sufficiently broad. If they are, then the government will often arrive at the conclusion that open code is preferable to proprietary code. At the very least, such an approach would lead to the conclusion that the government should not allow software patents to tilt the competitive horizon against open code projects.

the Reiser File System is sponsored primarily by the Defense Advanced Research Projects Agency (DARPA) and will be licensed under the GPL. For further details, see Evans and Reddy, "Government Preferences for Promoting Open-Source Software."

44. For example, the Patent and Trademark Law Amendments Act (also known as the Bayh-Dole Act) of 1980 encouraged universities and small businesses to commercialize inventions by permitting exclusive licensing of intellectual property that was developed with public funding. In exchange for the right to elect title to an invention, the licensor must agree to properly manage the invention and provide reports to the government. Since the passage of Bayh-Dole, universities have increasingly set up technology transfer programs and actively patented and commercialized inventions. See Council of Government Relations, "The Bayh-Dole Act: A Guide to the Law and Implementing Regulations," September 1999 (www.cogr.edu/bayh-dole.htm [May 20, 2002]).

45. For further discussion, see Evans and Reddy, "Government Preferences for Promoting Open-Source Software."

46. See presentation by James Bessen at the AEI-Brookings Joint Center for Regulatory Studies Conference "Is Open Source the Future of Software?" Washington, D.C., April 12, 2002.

Chapter 4
Open Source Baselines: Compared to What?

1. For example, Craig Mundie argues that General Public License software—as distinct from open source software—may lead to the "forking" of the code base (hence causing incompatibility), weakened interoperability, product instability, problematic strategic planning for business leaders, and the risk of forcing intellectual property into the public domain. Craig Mundie, "The Commercial Software Model," remarks at the Stern School of Business, New York University (www.microsoft.com/presspass/exec/craig/05-03sharedsource.asp [August 2002]).

2. "In the heat of their Hatfield-McCoy feud, Microsoft and Netscape have taken HTML hostage. Each company is proposing incompatible ways to extend the Web's lingua franca, exposing us to the danger that we'll soon have two different dialects." See Jesse Berst, "Microsoft, Netscape Feud Puts HTML's Future at Risk," April 11, 1997 (www.zdnet.com/anchordesk/story/story_827.html [August 2002]).

3. See Richard Stallman, "What Is Copyleft?" (www.fsf.org/copyleft/copyleft.html [August 2002]); Teresa Hill, "Fragmenting the Copyleft Movement: The Public Will Not Prevail," *Utah Law Review* (1999), p. 797.

4. See generally Richard Stallman, "Linux and the GNU Project" (www.gnu.org/gnu/linux-and-gnu.html [August 2002]).

5. There is an important quibble about what "modifies" means. The Free Software Foundation sometimes seems to suggest that any change in GPL-covered code would be a derivative work, thereby subject to copyright regulation. For a discussion of the derivation of subsequent distribution of a modified work as a form of original work versus the derivative work itself, see David McGowan, "Legal Implications of Open-Source Software," *Illinois Law Review* (2001), p. 254.

6. See, for example, Bjørn Reese and Daniel Stenberg, "Working without Copyleft," December 19, 2001 (www.oreillynet.com/pub/a/policy/2001/12/12/transition.html [August 2002]). The authors explain their feelings of repulsion for the GPL because, in its fear of corporate exploitation, it is uncooperative and has an overly extensive scope that becomes a deterrent to development.

7. "Apache soon became the number one server in the world. To this day, two-thirds of the servers on the World Wide Web are Apache servers." Lawrence Lessig, *The Future of Ideas* (Random House, 2001), pp. 55–56. See also David A. Wheeler, "Why Open Source Software/Free Software? Look at the Numbers!" (www.dwheeler.com/oss_fs_why.html [August 2002]).

8. "Source code is the code that programmers write. It is close to a natural language, but not quite a natural language. A program is written in source code, but to be run it must be converted into a language the machine can read. Some source code is converted on the fly—BASIC, for example, is usually interpreted by the computer as the computer runs a BASIC program. But most source code—or the most powerful source code—is 'compiled' before it is run. The computer converts the source code into either assembly code (which mavens can read) or object code (which only geniuses and machines can read). Object code is machine-readable. It is an undifferentiated string of 0s and 1s that instructs the machine about the tasks it is to perform. Programmers do not directly write object code, even if some are able to decipher it; programmers write source code. Object code speaks to the computer; source code speaks to humans and to computers (compilers); assembly code speaks to mavens and computers." See Lawrence Lessig, *Code and Other Laws of Cyberspace* (Basic Books, 1999), p. 103.

9. For a general introduction to public goods, see the discussion of common pool resources in Elinor Ostrom, *Governing the Commons: The Evolution of Institutions for Collective Action* (Cambridge University Press, 1990). She explores the nonexcludable nature of public goods via the thought-experiment of the grazing field "open to all," allowing unfettered access to all interested grazers (pp. 2–3). She also explains that, even in scenarios where excludability may be achieved, it is nevertheless practically impossible, since the costs of excluding others are prohibitive (pp. 30–33). As to the nonrivalrous nature of public goods, see Ostrom (p. 30) for a brief introduction to the notion of practically nonrivalrous public goods (allowance for maximal allocation of goods with no significant depletion of those goods for others), and see Lessig, *Future of Ideas*, pp. 20–23. (Language, for example, is a perfectly nonrivalrous public good, since your use of it doesn't impede mine.)

10. Ostrom, *Governing the Commons*, chap. 3. As Ostrom demonstrates, it doesn't follow that every public good requires state intervention for it to be supplied. Nor does it follow that every public good needs to be "privatized" for it to be supplied.

11. According to Yochai Benkler, "A commons-based information policy relies on the observation that some resources that serve as inputs for information production and exchange have economic or technological characteristics that make them susceptible to be allocated without requiring that any single organization, regulatory agency or property owner, clear conflicting uses of the resource. For example, the nonrivalrous nature of information, and the perfect renewability of radio frequency spectrum, create the possibility of sustainable commons in information used as an input into new information production, and in the RF spectrum, respectively." Benkler, "The Commons as a Neglected Factor of Information Policy," paper presented at the Twenty-sixth Annual Telecommunications Policy Research Conference, Alexandria, Va., October 1998 (www.law.nyu.edu/benklery/ [August 2002]). James Boyle describes the emerging quandary of how to entice development and creativity of information in the context of increasingly "free, complete, instantaneous, and universally available . . . information flows that are costless, general, and fast." Boyle, *Shamans, Software, and Spleens* (Harvard University Press, 1996), p. 35. David McGowan notes that "the low cost of copying and using code combined with the broad grants of the relevant licenses creates a situation that resembles a commons in some respects." McGowan, "Legal Implications," p. 244.

According to the UN secretary general's *Millennium Report* (p. 159): "[T]he core product in this sector—information—has unique attributes, not shared by others. The steel used to construct a building, or the boots worn by the workers constructing it, cannot be consumed by anyone else. Information is different. Not only is it available for multiple uses and users, it becomes more valuable the more it is used. The same is true of the networks that link up different sources of information." Reprinted in "Information as Global Public Good: A Right to Knowledge and Communication," Oxfam International Campaign Proposal 2000 (http://danny.oz.au/free-software/advocacy/oicampaign.html [August 2002]). From the viewpoint of one venture capitalist, "Put simply, in a world where there are essentially no costs to replicate content and it is effectively impossible to stop anyone from doing so at will, the current economic model underpinning content creation [including code writing] will be dead." See Dan Kohn, "Content Is a Pure Public Good" (http://db.tidbits.com/getbits.acgi?tbart=06604 [August 2002]). Finally, writing for the UN Food and Agriculture Organization, Bernard Woods advocates the transformation of digital technologies, including software and other facilitative technologies, from potential public goods into public goods in an almost traditional sense, that is, provided and guaranteed by governmental entities in a manner similar to utilities or a lighthouse. "A Public Good, a Private Responsibility" (www.fao.org/waicent/faoinfo/sustdev/dodirect/doengb02.htm [August 2002]).

12. See William M. Landes and Richard A. Posner, "An Economic Analysis of Copyright Law," *Journal of Legal Studies*, vol. 18, no. 2 (1989), pp. 325–63. Landes and Posner put forth an economic analysis of how the law may facilitate the maximally efficient design and enforcement of copyright law, stipulating that "for copyright law to promote economic efficiency, its principal legal doctrines must, at least approximately, maximize the benefits from creating additional works minus both the losses from limiting access and the costs of administering copyright protection."

13. See Kenneth J. Arrow, "Economic Welfare and the Allocation of Resources for Invention," in Richard R. Nelson, ed., *The Rate and Direction of Inventive Activity* (National Bureau of Economic Research and Princeton University Press, 1962), p. 609. For a more recent discussion of Arrow's analysis, see Gillian K. Hadfield, "The Economics of Copyright: An Historical Perspective," *ASCAP Copyright Law Symposium*, no. 38 (1992), pp. 39–40. Hadfield states that "when a resource is indivisible, the marginal cost of increasing the quantity available is zero; optimal allocation therefore requires that it be distributed at a price of zero. Yet clearly if price equals zero, then the fixed cost of producing the resource will not be covered by the market. . . . As in the case of ordinary public goods, information is underproduced to the extent that price is reduced to zero away from the average cost of its production."

14. However, tools to defeat some of the public goods aspects of software cannot eliminate them. There is certainly plenty of software piracy, despite the mix of public and private tools to protect against improper use.

15. "Once the work is created, the author's efforts can be incorporated into another copy virtually without cost." See Landes and Posner, "An Economic Analysis," p. 327.

16. This does not mean that there is always, or even often, a line dividing the necessary from the not. But where there has been protection sufficient to induce the production of a certain good, at least the terms of that protection should not be extended. See brief of "17 Economists" as amici curiae in support of petitioners to the Supreme Court of the United States, *Eldred* v. *Ashcroft* (www.eldred.cc/legal/supremecourt.html [August 2002]).

17. Most of the commentary on free and open source software has been "focused on explaining the rationale behind growing participation in this movement." See Siobhan Clare O'Mahony, "The Emergence of a New Commercial Actor," Ph.D. dissertation, Stanford University, 2002, p. 52, citing Josh Lerner and Jean Tirole, "The Simple Economics of Open Source," Working Paper W7600 (Cambridge, Mass.: National Bureau of Economic Research, 2002). See also Justin Pappas Johnson, "Economics of Open Source Software," working paper, May 17, 2001 (http://opensource.mit.edu/papers/johnsonopensource.pdf [August 2002]).

18. As William Baumol writes, "Individuals who have invested directly or indirectly in the economy's innovation processes can be estimated, conservatively, to obtain less than 10% of the total economic benefits contributed by new technology and new products." Baumol, "Pareto Optimal Sizes of Innovation

Spillovers" (www.econ.nyu.edu/user/baumolw/inovspi1.htm [August 2002]). Baumol's work is consistent with the extensive research of Eric von Hippel tracking the source of innovation in a wide range of contexts. As von Hippel has demonstrated, there are many contexts beyond software where innovation is provided by innovators who do not recover the value of the innovation. See Eric von Hippel, "Horizontal Innovation Networks—by and for Users" (http://opensource.mit.edu/papers/vonhippel3.pdf [August 2002]).

19. According to their websites, Peet's Coffee opened in Berkeley in 1966, while Starbucks opened in Seattle in 1971.

20. For a more skeptical view of this comparison, see Mathias Strasser, "A New Paradigm in Intellectual Property Law? The Case against Open Sources," *Stanford Technology Law Review* (2001), p. 4, para. 80.

21. See Carolyn Elefant, "Do Not Copy That Brief," *Legal Times Intellectual Property Magazine*, May 7, 2001 (www.his.com/~israel/loce/press.html [August 2002]). Elefant argues that although legal briefs appear to satisfy the criteria traditionally demanded of copyrightable material (such as originality, inclusion in statutorily identified categories of the Copyright Act, fixation in tangible medium), whether lawyers may expect or should want briefs to be copyrightable is due further thought. For example, she argues that even if they are copyrightable, there are massive enforcement hurdles, and tightening of access to briefs may damage research interests and decrease quality of legal briefs.

22. "Conventional markets work very well where the commodities are well defined, where the demand and the nature of the customer's need are well defined, and where the property rights or contractual relationships are well defined. But software is not typically a standardized commodity. Packaged software has never represented as much of a third of all software investment in the United States. Open source is thus in some sense a better business model because it's easier to customize." James Bessen, remarks at the AEI-Brookings Joint Center for Regulatory Studies conference "Is Open Source the Future of Software?" April 12, 2002 (www.aei.brookings.org/events/page.php?id=59#bessen).

23. Baumol, "Pareto Optimal Sizes of Innovation Spillovers."

24. See www.linuxlinks.com/embedded/.

25. For a brief summary, see David S. Evans, "Is Free Software the Wave of the Future?" *Milken Institute Review*, 4th quarter (2001), pp. 38–39. The most significant reason for commercial entities to support open or free software projects is that they keep the cost of complements low. IBM sells hardware; if it can keep the cost of operating systems low, that will increase the demand for its hardware. See Lessig, *Future*, pp. 69–70.

26. See James Bessen, "Open Source Software: Free Provision of Complex Public Goods," ROI Working Paper, July 2002 (www.researchoninnovation.org/online.htm). For the argument that peer pressure increases quality in open systems, see Harlan D. Mills, "Top-down Programming in Large Systems," in Randall Rustin, ed., *Debugging Techniques in Large Systems* (Englewood Cliffs, N.J.: Prentice Hall, 1971).

27. My argument is consistent with a point long made by Eric von Hippel: firms that give users the opportunity to innovate upon their product do better in the market than those that do not. This gives firms a reason to support a kind of open source development, and this open source development is supported by innovators who never capture the full value of their innovation. See Eric von Hippel, "Customers as Innovators: A New Way to Create Value," *Harvard Business Review*, vol. 80, no. 4 (2002), p. 74.

28. For details on French policy, see "France towards Open e-Government—Government Agency to Enforce Open Standards and Promote Open Source/Free Software," November 21, 2001 (http://old.lwn.net/2001/1129/pr/pr4501.php3 [August 2002]). On Germany, see David McHugh, "German Government Signs Deal With IBM," AP Wire Service, June 3, 2002 (www.radicus.net/news/wed/cx/agermany-linux.rb-t_cu3.asp). On China, see Andy Tai, "Taiwan to Start National Plan to Push Free Software," June 3, 2002 (www.kuro5hin.org/story/2002/6/3/55433/41738 [August 2002]).

29. See David S. Evans and Bernard Reddy, "Government Preferences for Promoting Open-Source Software: A Solution in Search of a Problem," working paper (Cambridge, Mass.: National Economic Research Associates, May 21, 2002) (http://ssrn.com/abstract_id=313202 [August 2002]). This paper is a careful and extraordinarily complete analysis of the issue. While it asserts that I have argued that open source and free software is "innovative" (p. 64), in fact I have not made any claim about whether the software itself is innovative. My claim is that the platform it helped build produced innovation. Evans and Reddy collect a wide range of reasons why the government might, from an economic perspective, prefer open source or free software. They have not included the reasons provided here.

30. Mundie, "The Commercial Software Model."

31. "With respect to procurement, I would run the government like a business." See David Evans, remarks at the AEI-Brookings Joint Center for Regulatory Studies conference "Is Open Source the Future of Software?" April 12, 2002 (www.aei.brookings.org/events/page.php?id=59#evans).

32. Mundie, "The Commercial Software Model." Microsoft makes an independent point about the potential "danger" of GPL licensed for other intellectual property owned by the software developer. See also www.microsoft.com/sharedsource.

33. Evans and Reddy make a similar argument against the government funding GPL-covered software. Evans and Reddy, "Government Preferences," pp. 74–76. They note that the government has traditionally funded research that either goes into the public domain or is used by the military or is spun off for commercial purposes (p. 75). But when the government funds research that produces GPL-covered software, Evans and Reddy contend that "there is no economic justification for this support of the GPL." Their argument is in essence that "support of GPL projects is incompatible with commercial spin-off efforts, since the GPL is incompatible with proprietary, commercial software" (p. 76).

There are many unstated assumptions built into this argument. First, to say that the GPL is "incompatible with proprietary, commercial software" is not to say it is "incompatible with commercial spin-off efforts." There are plenty of commercial firms that develop and support GPL software—IBM to name just one. No doubt GPL-covered code is not a resource for "proprietary, commercial software" development, to the extent that development modifies and redistributes GPL-covered code. But again, there is plenty of proprietary, commercial software development that need not modify and redistribute GPL-covered code, such as GNU/Linux, just as there is plenty of proprietary, commercial software development that need not modify and redistribute Windows XP.

Second, this argument takes the commercialization of government-funded research as the baseline and measures "economic justification" against that. But there is no argument for this baseline. Certainly the government funds lots of research that passes into the public domain. Evans and Reddy have not provided an argument against that. To the extent innovation passes into the public domain, it may or may not be less exploitable by commercial entities. It was a concern about a lack of incentives to exploit resources that lead to the passage of the Bayh-Dole Act, permitting the patenting of government-funded research. The effect of that act on research has been extremely controversial; for example, see Arti Kaur Rai, "Regulating Scientific Research: Intellectual Property Rights and the Norms of Science," *Northwestern University Law Review*, vol. 94 (1999), pp. 136–37. But unless there is some reason to reject supporting research for the public domain, there cannot be a general, economic reason to reject supporting research for the GPL. The only general argument is the one Microsoft seems to be making. But this again would apply to government support of proprietary software development as much as government support of GPL software development.

34. See Lessig, *Future*, pp. 206–14.

Chapter 5 The Future of Software: Enabling the Marketplace to Decide

1. Although these views are often associated with the Free Software Foundation and its founder, Richard Stallman, strands of this thinking can be found throughout the open source community.

2. This topic is discussed in more detail in a subsequent section. "GNU" is a recursive acronym that stands for "GNU's Not UNIX." The acronym reflects the original purpose of the GNU Project, which was to develop a "free" alternative to the UNIX operating system. See Richard Stallman, *The GNU Project* (www.gnu.org [2002]).

3. For a description of the BSD license and similar licenses as "permissive," see Sean Doherty, "The Law and Open Source Software," *Network Computing*, October 29, 2001.