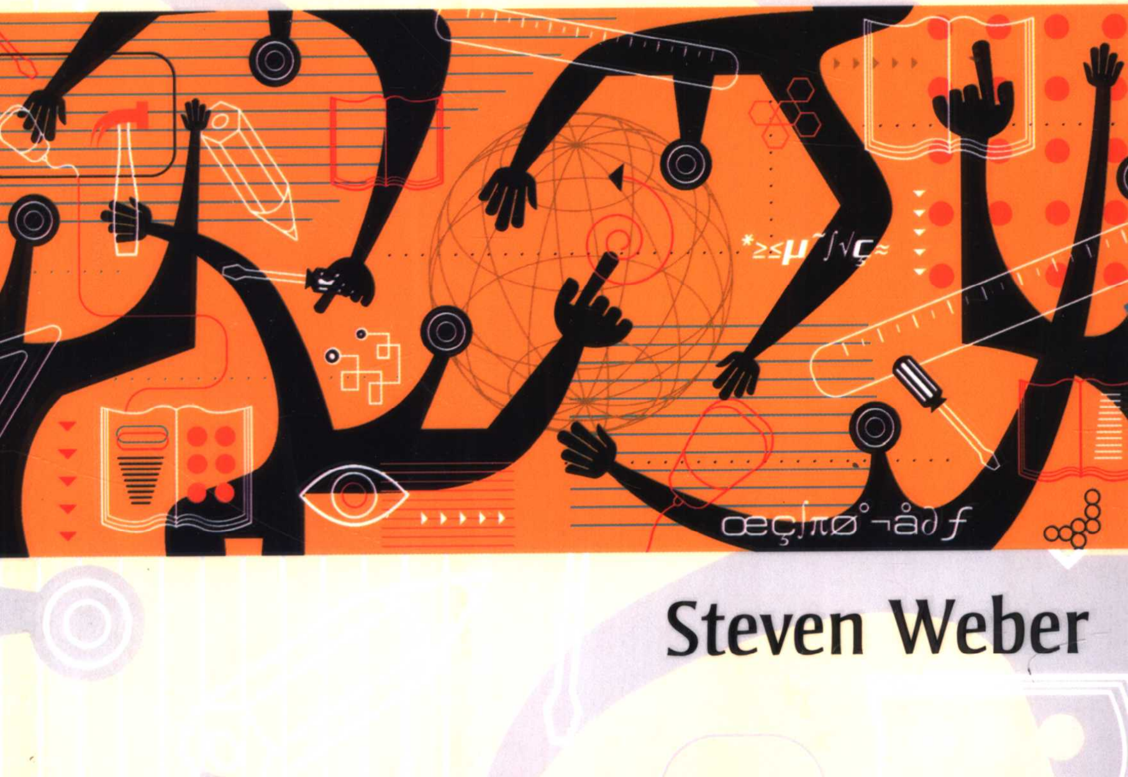
THE SUCCESS OF OPEN SOURCE



Steven Weber

ovo
D919
248

STEVEN WEBER

The Success of Open Source

HARVARD UNIVERSITY PRESS
Cambridge, Massachusetts, and London, England
2004

11

Library of Congress Cataloging-in-Publication Data

Weber, Steven, 1961–
The success of open source / Steven Weber.
p. cm.
Includes bibliographical references and index.
ISBN 0-674-01292-5 (alk. paper)
1. Open source software. I. Title.

QA76.76.S46W43 2004
005.3—dc22 2003056916

Preface

Several years ago when I began thinking about open source software, I had to convince just about everyone I talked to, outside of a narrow technology community, that this was a real phenomenon and something worth studying in a serious way. I no longer have to make that case. Open source has become a subject of real interest to a wide-ranging swath of people and organizations. I hope this book will explain to many of those people how the open source process works and why the success of open source is broadly significant.

I'm a political scientist and I worry more about how communities are governed than I do about technology per se. I became interested in open source as an emerging technological community that seemed to solve what I see as very tricky but basically familiar governance problems, in a very unfamiliar and intriguing way. In the end I've decided, and I argue in this book, that the open source community has done something even more important. By experimenting with fundamental notions of what constitutes property, this community has reframed and recast some of the most basic problems of governance. At the same time, it is remaking the politics and economics of the software world. If you believe (as I do) that software constitutes at once some of the core tools and core rules for the future of how human beings work together to create wealth, beauty, new ideas, and solutions to problems, then understanding how open source can change those processes is very important.

I had a huge amount of help and support in writing this book. The

University of California at Berkeley, the Ford Foundation, the Markle Foundation, the Social Science Research Council, the Institute on Global Conflict and Cooperation, and University of California's Industry-University Cooperative Research Program made bets that I would produce something valuable. My close friends and colleagues at the Berkeley Roundtable on the International Economy, Global Business Network, and the Monitor Group made their bets as well but more importantly gave me the gift of great ideas and probing questions. More people than I can possibly remember from many different walks of life read versions of chapters, talked me through sticky problems, and put up with my sometimes naïve questions. I owe particularly large thanks to Jonathan Aronson, Michael Barnett, David Bollier, Denise Caruso, Peter Cowhey, Jerome Feldman, Brad DeLong, Rana Nanjappa, Elliot Posner, AnnaLee Saxenian, Janice Stein, Mark Stone, Nick Ziegler, John Zysman, and a group of the best graduate students in the world. And a special thanks to Michael Aronson and Elizabeth Collins from Harvard University Press.

And of course I owe the greatest thanks to the open source communities I have studied and to the people who contribute their time and energy to that work. I know you weren't trying to build a huge data set for some researcher to come in and understand how you were doing what you were doing. You are just trying to build the best software you can in the way that makes sense to you. I hope I've done some justice to that effort in my attempts to understand what makes open source succeed and why that success matters as much as I think it does.

I want to dedicate this book to my three life partners, who helped me through every day I worked on it. Even though only one of them walks on two legs.

Contents

CHAPTER 1

Property and the Problem of Software

This is a book about property and how it underpins the social organization of cooperation and production in a digital era. I mean "property" in a broad sense—not only who owns what, but what it means to own something, what rights and responsibilities property confers, and where those ideas come from and how they spread. It is a story of how social organization can change the meaning of property, and conversely, how shifting notions of property can alter the possibilities of social organization.

I explain the creation of a particular kind of software—open source software—as an experiment in social organization around a distinctive notion of property. The conventional notion of property is, of course, the right to exclude you from using something that belongs to me. Property in open source is configured fundamentally around the right to distribute, not the right to exclude. If that sentence feels awkward on first reading, that is a testimony to just how deeply embedded in our intuitions and institutions the exclusion view of property really is.

Open source is an experiment in building a political economy—that is, a system of sustainable value creation and a set of governance mechanisms. In this case it is a governance system that holds together a community of producers around this counterintuitive notion of property rights as distribution. It is also a political economy that taps into a broad range of human motivations and relies on a creative and evolving set of organizational structures to coordinate behavior. What would a broader version of this political economy really look like? This book uses the open source story as a vehicle for proposing a set of preliminary answers to that very large question.

The way in is to answer two more immediate questions about open source. How is it that groups of computer programmers (sometimes very large groups) made up of individuals separated by geography, corporate boundaries, culture, language, and other characteristics, and connected mainly via telecommunications bandwidth, manage to work together over time and build complex, sophisticated software systems outside the boundaries of a corporate structure and for no direct monetary compensation? And why does the answer to that question matter to anyone who is not a computer programmer?

Let me restate these questions as an observation and two general propositions that together provoked me to write this book. The observation is that collaborative open source software projects such as Linux and Apache have demonstrated that a large and complex system of software code can be built, maintained, developed, and extended in a nonproprietary setting in which many developers work in a highly parallel, relatively unstructured way. The first proposition is that this is an important puzzle for social scientists worrying about problems of both small- and large-scale cooperation (which is just about every social scientist, in one way or another). It is also an important puzzle for anyone who struggles, in theory or in practice, with the limits to very complex divisions of labor and the management of knowledge in that setting.

The second proposition is that the open source software process is a real-world, researchable example of a community and a knowledge production process that has been fundamentally changed, or created in significant ways, by Internet technology. Understanding the open source process can generate new perspectives on very old and essential problems of social cooperation. And it can provide an early perspective on some of the institutional, political, and economic consequences for human societies of the telecommunications and Internet revolutions.

This book explains how the open source software process works. It is broadly a book about technology and society, in the sense that changes in technology uncover hidden assumptions of inevitability in production systems and the social arrangements that accompany them. It is also about computers and software, because the success of open source rests ultimately on computer code, code that people often find more functional, reliable, and faster to evolve than most proprietary software built inside a conventional corporate organization. It is a business and legal story as well. Open source code does not obliterate profit, capitalism, or intellectual property rights. Companies and individuals are creating intellectual products and making money from open source software code, while inventing new business models and notions about property along the way.

Ultimately the success of open source is a political story. The open source software process is not a chaotic free-for-all in which everyone has equal power and influence. And it is certainly not an idyllic community of like-minded friends in which consensus reigns and agreement is easy. In fact, conflict is not unusual in this community; it's endemic and inherent to the open source process. The management of conflict is politics and indeed there is a political organization at work here, with the standard accoutrements of power, interests, rules, behavioral norms, decision-making procedures, and sanctioning mechanisms. But it is not a political organization that looks familiar to the logic of an industrial-era political economy.

The Analytic Problem of Open Source

Think of a body of software code as a set of instructions for a computer—an artifact, a "thing" in and of itself. In that context, what is open source software and how is it different from the proprietary software products that companies like Microsoft and Oracle build and sell?

Consider a simple analogy to Coca-Cola.[1] Coca-Cola sells bottles of soda to consumers. Consumers use (that is, drink) the soda. Some consumers read the list of ingredients on the bottle, but that list of ingredients is surprisingly generic. Coca-Cola has a proprietary formula that it will not divulge, on the bottle or anywhere else. This formula is the knowledge that makes it possible for Coke to combine sugar, water, and a few other readily available ingredients in particular proportions with a secret flavoring mix and produce something of great value. The point is that the bubbly liquid in your glass cannot be reverse-engineered into its constituent parts. You can buy Coke and you can drink it, but you can't *understand* it in a way that would let you reproduce the drink, or improve upon it and distribute your cola drink to the rest of the world.

Standard economics of intellectual property rights provides a straightforward account of why the Coca-Cola production regime is organized this way. The core problem of intellectual property is supposed to be about creating incentives for innovators. Patents, copyrights, licensing schemes, and other means of "protecting" knowledge ensure that economic rents are created and that some proportion of those rents can be appropriated by the innovator. If that were not the case, a new and improved formula would be immediately available in full and for free to anyone who chose to look at it. The person who invented the formula would have no special and defensible economic claim on a share of the profits that might be made by selling drinks engineered from the innovation. And so the system unravels, because that person no longer has any rational incentive to innovate in the first place.

The production of computer software is typically organized under a similar regime, with a parallel argument behind it. You can buy Microsoft Windows and you can use it on your computer, but you cannot reproduce it, modify it, improve it, and redistribute your own version of Windows to others. Copyright, licenses, patents, and other legal structures provide a layer of legal protection to this regime, but there is an even more fundamental mechanism that stops you from doing any of these things. Just as Coca-Cola does not release its formula, Microsoft and other proprietary software makers do not release their source code.

Source code is a list of instructions that make up the "recipe" for a software package. Software engineers write source code in a programming language (like C++ or FORTRAN) that a human can read and understand, as well as fix and modify. Most commercial software is released in machine language or what are called "binaries"—a long string of ones and zeros that a computer can read and execute, but a human cannot read.[2] The source code is basically the recipe for the binaries; and if you have the source code, you can understand what the author was trying to accomplish when she wrote the program—which means you can modify it. If you have just the binaries, you typically cannot either understand or modify them. Therefore, shipping binary code is a very effective way for proprietary software companies to control what you can do with the software you buy.

Proprietary source code is the touchstone of the conventional intellectual property regime for computer software. Proprietary source code is supposed to be the fundamental reason why Microsoft can sell Windows for around $100 (or why Oracle can sell its sophisticated data management software for many thousands of dollars) and distribute some of that money to programmers who write the code—and thus provide incentives for them to innovate.

Open source software simply inverts this logic. The essence of open source software is that source code is free. That is, the source code for open source software is released along with the software to anyone and everyone who chooses to use it. "Free" in this context means freedom (not necessarily zero price). Free source code is open, public, and nonproprietary. As Richard Stallman puts it, freedom includes the right to run the program for any purpose, to study how it works and adapt it to your own needs, to redistribute copies to others, and to improve the program and share your improvements with the community so that all benefit.[3] Programmers often explain it with simple shorthand: when you hear the term free software, think "free speech" not "free beer." Or, in pseudo-French, software libre not software gratis.

The core of this new model is captured in three essential features of the semiofficial "Open Source Definition":

- Source code must be distributed with the software or otherwise made available for no more than the cost of distribution.
- Anyone may redistribute the software for free, without royalties or licensing fees to the author.
- Anyone may modify the software or derive other software from it, and then distribute the modified software under the same terms.[4]

If you array these terms against the conventional intellectual property story for software, open source software really should not exist. Or at best it should be confined to small niches outside the mainstream information technology economy, perhaps among a tightly bound group of enthusiastic hobbyists who create and share source code for the love of the challenge.

Here's the empirical problem: Open source software is a real, not marginal, phenomenon. It is already a major part of the mainstream information technology economy, and it increasingly dominates aspects of that economy that will probably be the leading edge (in technological and market terms) over the next decade. There exist thousands of open source projects, ranging from small utilities and device drivers to office suites like OpenOffice, database systems like MySQL, and operating systems like Linux and BSD derivatives.[5] Linux and Apache attract the most public attention. Apache simply dominates the web server market—over 65 percent of all active web sites use Apache.[6] Nearly 40 percent of large American companies use Linux in some form; Linux is the operating system for more than a third of active web servers and holds almost 14 percent of the large server market overall.[7]

Sendmail is an open source email transfer and management program that powers about 80 percent of the world's mail servers. BIND is an open source program that acts as the major addressing system for the Internet. If you use Google to search the web, you use a cluster of 10,000 computers running Linux. Yahoo! runs its directory services on FreeBSD, another open source operating system. If you saw the movies *Titanic* or *Lord of the Rings*, you were watching special effects rendered on Linux machines that are running at companies like Disney, DreamWorks, and Pixar. Increasingly, open source software is running major enterprise applications for large and small corporations alike. Amazon, E*Trade, Reuters, and Merrill Lynch are examples of companies that have recently switched backend computer systems to Linux. Large parts of the U.S. government, including the Defense Department, the Department of Energy, and the National Security Agency, work with open source software. National, state, and municipal governments from Germany to Peru to China are considering and in some cases mandating the use of open source software for e-government applications. IBM is now a major champion of open source after publicly declaring in 2001 a $1 billion commitment to developing technology and recasting central parts of its business models around Linux and other open source programs. Hewlett-Packard, Motorola, Dell, Oracle, Intel, and Sun Microsystems have all made serious (if less radical) commitments to open source software.

The fact that Linux is probably not running your desktop computer today does not diminish the significance of what is happening with open source. That is partly because more PCs and computing appliances will run Linux and open source programs in the next few years.[8] But Windows on your desktop is not important for a more fundamental reason, and that is because your PC desktop is becoming much less important. Even Microsoft knows and acknowledges this—that recognition is at the heart of the company's move toward web services and the "dot-net" architecture. Sun Microsystems claimed a long time ago that "the network is the computer" and the technology is upholding that claim. Your desktop is like the steering wheel to your car—important, but not nearly as important as the engine. The engine is the Internet, and it is increasingly built on open source software.

Computer scientists and software engineers value Linux and other open source software packages primarily for their technical characteristics. But as open source has begun over the last several years to attract more public attention, it has taken on a peculiar mantle and become a kind of Internet era Rorschach test. People often see in the open source software movement the politics that they would like to see—a libertarian reverie, a perfect meritocracy, a utopian gift culture that celebrates an economics of abundance instead of scarcity, a virtual or electronic existence proof of communitarian ideals, a political movement aimed at replacing obsolete nineteenth-century capitalist structures with new "relations of production" more suited to the Information Age.

It is almost too easy to criticize some of the more lavish claims. Like many things about the Internet era, open source software is an odd mix of overblown hype and profound innovation. The hype should be at least partly forgiven. The open source phenomenon is in some ways the first and certainly one of the most prominent indigenous political statements of the digital world. Unlike the shooting star that was Napster, the roots of open source go back to the beginning of modern computing; it is a productive movement intimately linked to the mainstream economy; and it is developing and growing an increasingly self-conscious identification as a community that specifies its own norms and values.

Some of those values sound extraordinarily compelling, particularly when compared to darkly dystopic visions of the Internet-enabled society as one in which computer code leads to a radically privatized, perfectly regulated, tightly controlled world in which technology enforces upon the many the shape of a market that is preferred by and benefits the few. In the widely read book *Code and Other Laws of Cyberspace*, Lawrence Lessig repeatedly invokes the idea of open source as a major challenge and counterpoint to the possibilities for government and corporate control of the architecture that will help shape the e-society. He implies that this is part of an almost epochal battle over who will control what in the midst of a technological revolution, and that open source is on the right side of that battle.[9] Lessig is hardly alone in this view.[10] And it is an important point to make, although I will show that the situation is considerably more complicated than "open= good, closed=bad."[11] To get to a more nuanced understanding of what is at stake, we first should confront in detail the problem of how open source comes to be, what its boundaries and constraints are, what makes it work as a social and economic system, and what that system in turn makes possible elsewhere. That is the purpose of this book.

The Political Economy of Open Source

My starting point for explaining the open source process is the lens of political economy. I will situate the puzzle to start in modern concepts from political economy and then say more precisely why open source challenges some conventional theories about the organization of production, and how it affects and is affected by society. This lens represents a choice: There are other starting points you could choose; and the choice does matter in terms of where you come out as well as where you start. One of the strengths of the political economy perspective in fact is that it can naturally open up to a much broader set of discussions, and I will do so particularly in the conclusion to the book. The point is to take the political economy perspective as a useful focusing device for a discussion of a very complex set of human and social behaviors.

One of the foundational problems of political economy is collective action. People do not easily work together in large groups toward a joint goal. There are many reasons for this: People have different preferences around the goal, they have different tolerances for costs and effort, they find it difficult to evaluate the importance of others' and their own contributions, and in many cases they would come out better if they were able to sit back and allow somebody else to contribute in their place. The classic modern statement of the problem is Mancur Olson's book *The Logic of Collective Action.* Olson's arguments have been refined over time, but the core logic has become almost the equivalent of an instinct for people who think about politics and organization. And thus the natural attraction of the open source process to this conceptual frame: Intuition tells us that thousands of volunteers are unlikely to come together to collaborate on a complex economic project, sustain that collaboration over time, and build something that they give away freely, particularly something that can beat some of the largest and richest business enterprises in the world at their own game.

Marc Smith and Peter Kollock took that intuition a step further when they wrote about Linux as "the impossible public good."[12] Linux is nonrival and nonexcludable. Anyone can download a copy of Linux along with its source code for free, which means it is truly nonexcludable. And because it is a digital product that can be replicated infinitely at zero cost, it is truly nonrival. For well-known reasons that track with the intellectual property rationale, public goods tend to be underprovided in social settings. In other words, it is hard for a community of human beings to organize and sustain organization for the production and maintenance of public goods. The situation with Linux ought to be at the worse end of the spectrum of public goods because it is subject additionally to "collective provision." In other words, the production of this particular good depends on contributions from a large number of developers. Stark economic logic seems to undermine the foundations for Linux and thus make it impossible.

The elementary political economy question about open source software is simple. Why would any person choose to contribute—voluntarily—to a public good that she can partake of, unchecked, as a free rider on the effort of others? Because every individual can see that not only her own incentives but the incentives of other individuals are thus aligned, the system ought to unravel backward so no one makes substantial contributions, and the good never comes to be in the first place.

But Linux is also an impossibly complex good. An operating system is a huge, complicated, intricate piece of code that controls the basic, critical functions of a computer. Everything depends on it. It is the platform on which applications—be they word processors, spreadsheets, databases, or anything else—sit and run. To design a robust operating system and to implement that design in software code is a gargantuan task. Testing, debugging, maintaining, and evolving the system over time are even harder. Computer users will run an operating system in a nearly infinite number of settings, with functionally infinite permutations of behavior, leading to infinite possible paths through the lines of code. Complex software is not like a book, even the longest and most complex book ever written. It is more like a living organism that must continually adapt and adjust to the different environments and tasks that the world puts in front of it.

There was a time when a single determined individual could write the core of a simple operating system for a primitive computer. But given the demands of computer applications and the capabilities of hardware technology at present, that is no longer conceivable. The task needs to be divided somehow. This immediately raises a second core political economy question, about coordination of a division of labor. The standard answer to this question has been to organize labor within a centralized, hierarchical structure—that is, a firm. Within the firm an authority can make decisions about the division of labor and set up systems that transfer needed information back and forth between the individuals or teams that are working on particular chunks of the project. The boundaries of the firm are determined by make-or-buy decisions that follow from the logic of transaction cost economics. The system manages complexity through formal organization and explicit authority to make decisions within the firm as well as price coordination within markets between firms.[13]

Even this caricatured model of industrial-era organization for production is hardly perfect. It is expensive and sometimes awkward to move information and knowledge around, to monitor the actions of labor, and to enforce decisions on individuals. No one says that hierarchical coordination in a complex production task like software development is efficient, only that it is less inefficient than the alternatives. And it does seem to work at some level. Within companies, the job gets done and complex software—imperfect, buggy, and expensive, but functional—does get produced. And thus a third core political economy question arises: Is this an inevitable way of organizing the production process for software (and, perhaps by implication, other complex knowledge goods)? Is it the best way?

Eric Raymond, computer hacker turned unofficial ethnographer of the open source movement, draws a contrast between cathedrals and bazaars as icons of organizational structure. Cathedrals are designed from the top down, then built by coordinated teams who are tasked by and answer to a central authority that implements a master plan. The open source process seems to confound this hierarchical model. Raymond sees instead a "great babbling bazaar of different agendas and approaches."[14] Yet this bazaar has produced software packages that develop "from strength to strength at a speed barely imaginable to cathedral builders."[15]

There is some hyperbole here, and the imagery of chaos and invisible hands in the bazaar misleads by distracting attention from what are the real organizational structures within open source. But focus for the moment on Raymond's core observation. Many computer programmers believe that Linux and other open source software packages have evolved into code that is superior to what hierarchical organizations can produce. The quality of software is to some degree a subjective judgment; and like "good art," a lot depends on what you want to do with the software and in what setting. But the technical opinions are serious ones. Ultimately, so are the opinions expressed in market share, and particularly in the success of open source software in taking away market share from proprietary alternatives.

To summarize and set the problem, open source poses three interesting questions for political economy:

- *Motivation of individuals:* The microfoundations of the open source process depend on individual behavior that is at first glance surprising, even startling. Public goods theory predicts that nonrival and nonexcludable goods ought to encourage free riding. Particularly if the good is subject to collective provision, and many people must contribute together to get something of value, the system should unravel backward toward underprovision. Why, then, do highly talented programmers choose voluntarily to allocate some or a substantial portion of their time and mind space to a joint project for which they will not be compensated?
- *Coordination:* How and why do these individuals coordinate their contributions on a single focal point? The political economy of any production process depends on pulling together individual efforts in a way that they add up to a functioning product. Authority within a firm and the price mechanism across firms are standard means of coordinating specialized knowledge in a