Steven Weber **The Success of Open Source**

Forthcoming 2003, Harvard University Press

Chapter 1: Introduction

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, it 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 hold 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 this political economy really look like? This book uses the open source story as a window to propose 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, empirically, that a large complex system of software code can be built, maintained, developed, and extended in a non-proprietary setting where many developers work in a highly parallel, relatively unstructured way. The first proposition is that this is an important puzzle for social scientists who think about problems of small and large-scale cooperation (which is just about every social scientist, in one way or another). 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. To understand 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 revolution(s).

This book is the story of how the open source software process works. It is partly a technical story, because the success of open source rests ultimately on the computer code, code that people find more reliable and faster to evolve than most proprietary software built inside a more conventional corporate organization. It is also a business and legal story. Open source code does not obliterate profit, capitalism or the general concept of intellectual property rights. Companies and individuals are creating intellectual products and making money from open source software code, 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 where everyone has equal power and influence. And it is certainly not an idyllic community of like-minded friends where consensus reigns and agreement is easy. In fact, conflict is not unusual in this community; it's endemic and in

a real sense 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 industrial era political economy.

The Analytic Problem of Open Source

Think of a body of software code as a set of instructions for a general purpose 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 may choose to read the list of ingredients on the bottle. But that list of ingredients is surprisingly generic. Coca-cola has a proprietary 'formula' that it does not and will not tell you about, on the bottle or anywhere else. The 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 some '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 empower you to reproduce the drink, or improve upon it and distribute your improved 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 assure 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

[1] Mitchell Stoltz suggested this analogy to me.

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' and make any sense of.[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 he or she wrote a program – which means you can modify it. If you just have the binaries, you almost always cannot. Which means that shipping binary code to run on your machine generally is a very effective way for proprietary software companies to control what you can and cannot 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 System 8 data management software for many thousands of dollars) and

[2] More precisely, programmers write programs in source code. Computers execute instructions in machine language. A separate program called a compiler transforms source code to object code that is either directly executable or undergoes another step into machine language.

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 as I will explain in a later chapter). Free source code is open, public, and non-proprietary. 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 "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, then 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. (I will come back to these arguments in much more detail later.)

[3] Free Software Foundation, "The Free Software Definition," http://www.fsf.org/philosophy/free-sw.html, Boston, MA: Free Software Foundation, Inc., 1996.
[4] See the full definition at http://www.opensource.org/docs/definition.php (Open Source Initiative, "The Open Source Definition, Version 1.9," 2002)

Here's the empirical problem: open source software is a real not just marginal phenomenon. It is already a major part of the mainstream information technology economy and it increasingly dominates aspects of that economy that will likely be the leading edge (in technological and market terms) over the next decade.

Insert one or two paragraphs with latest evocative examples and updated numbers as close to publication as possible, to replace this one. 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% of all active web sites use Apache.[6] Linux is the operating system for more than a third of active web servers. Sendmail is an open source email transfer and management program that powers about 80% 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 are using 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 US 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 business models around Linux and other open source programs. Hewlett Packard, Motorola, other recent converts….

[5] The open source software repository Sourceforge (www.sourceforge.org) held over 40,000 projects as of August 2002.
[6] http://www.netcraft.com/survey/

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 pc's and computing appliances will in fact run Linux and open source programs in the next few years.[7] 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 said a long time ago that 'the network is the computer' and the technology is proving that to be correct. 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 19th 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 (and indeed I will do so later). 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. It comes in part because 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 particular norms and values.

[7] If you use a Macintosh you are already there: OS X is built on open source code.

Some of those values sound extraordinarily compelling, particularly when compared to darkly dystopic visions of the Internet-enabled society as one where computer code leads to a radically privatized, perfectly regulated, tightly government and/or corporation – controlled world where 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 shall control what in the midst of a technological revolution, and that open source is on the right side of that battle.[8] Lessig is hardly alone in this view.[9] And it is an important point to make, although I will show that the situation is considerably more complicated than 'open=good, closed=bad'.[10] 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 to explain 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 one 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

---

[8] Lawrence Lessig, Code and Other Laws of Cyberspace, New York: Basic Books, 2000; and Lessig, "Open Code and Open Societies: Values of Internet Governance," Chicago-Kent Law Review 74. 1999. pp. 101-116.

[9] other cites: does Andrew Shapiro say it in control revolution? Get some other examples there are tons Boyle. Bollier.

[10] Peter Schwartz, Peter Leyden, Joel Hyatt, The Long Boom: A Vision for the Coming Age of Prosperity, Cambridge, MA: Perseus Publishing, 1999.

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 contribution, 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.'[11] Linux is non-rival and non-excludable. Anyone can download a copy of Linux along with its source code for free, which means it is truly non-excludable. And because it is a digital product that can be replicated infinitely at zero cost, it is truly non-rival. For well-known reasons that track with the intellectual property rationale I recounted earlier, 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 since 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.

[11] Marc A. Smith and Peter Kollock, eds. Communities in Cyberspace. London: Routledge, 1999, p. 230.

The elementary political economy question about open source software is simple. Why would any particular person choose to contribute – voluntarily – to a public good that she can partake of, unchecked, as a free-rider on the effort of others? Since every individual can see that not only her own incentives but the incentives of other individuals are thus aligned, the system ought to unravel backwards so that 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, maintenance, and evolving the system over time is 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 really 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 (indeed Chapter 2 recounts this). 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 makes decisions about the division of labor, who will do what, and sets 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 a 'make' or 'buy' decision that follows from the logic of transaction cost economics. The system

manages complexity through formal organization and explicit authority to make decisions within the firm, and price coordination within markets between firms.[12]

Even this cartoon 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, 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 alternatives. And it does seem to work – 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: 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."[13] Yet this bazaar has produced software packages that develop 'from strength to strength at a speed barely imaginable to cathedral builders."[14]

There is some hyperbole here, and I will question in later chapters the bazaar icon because I think imagery of chaos and invisible hands 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.

---

[12] Of course organization theorists know that a lot of management goes on 'informally' within the interstices of this structure, but the structure is still there to make it possible.

[13] Eric Raymond, "The Cathedral and the Bazaar," in The Cathedral and the Bazaar: Musings On Linux and Open Source by an Accidental Revolutionary. Sebastopol CA: O'Reilly Publishing, 1999, p. 30.

[14] Raymond, "The Cathedral and the Bazaar,", p. 30.

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 micro-foundations of the open source process depend on individual behavior that is at first glance surprising, even startling. Public goods theory predicts that non-rival and non-excludable goods ought to encourage free riding. Particularly if the good is subject to collective provision, where many people must contribute together in order to get something of value, the system should unravel backwards toward 'underprovision'. Why, then, do highly talented programmers choose voluntarily to allocate some or a substantial portion of their time and mind-space (that are both limited and valuable resources) 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 to coordinate specialized knowledge in a highly differentiated division of labor, but neither is operative in open source. Instead, individuals choose for themselves what they want to work on. Money is not a central part of the equation. And any individual can freely modify source code then redistribute modified versions to others. A simple analogy to ecology suggests what might happen over time as modifications accumulate along different branching chains. Speciation – what computer scientists call code-forking – seems likely. In effect the system evolves into incompatible versions. Synergies in development are lost. And any particular developer has to choose one or another version as the basis for her future work. As I will explain in Chapter 2 this is essentially what happened to another major operating system, UNIX, in the 1980s. How does the open source process sustain coordinated cooperation among a large number of contributors, outside the bounds of hierarchical or market mechanisms?

- **Complexity**: Software is an extraordinarily complex technical artifact. In The Mythical Man-Month, a classic study of the social organization of computer programming, Frederick Brooks noted that when large organizations add manpower to a software project that is behind schedule, the project typically falls even further behind schedule.[15] He explained this with an argument that is now known as Brooks' Law. Brooks' Law says that as you raise the number of programmers on a project, the work that gets done scales linearly, while complexity and vulnerability to mistakes scales geometrically. This is supposed to be inherent in the logic of the division of labor – the geometric progression represents the scaling of the number of possible communication paths and interfaces between pieces of code written by individual developers. Chapter 3 considers in detail the deeper line of reasoning behind this argument, which is actually an incredibly interesting statement about the relationship between complex systems of meaning and the imperfections of human communication. Recognize for the moment the challenge it poses to organization. What is the nature of governance within the open source process, that enables this community to manage the implications of this 'law' and perform successfully with such complex systems?

The book answers these questions by developing a multi-layered explanatory model of the open source process (the core of the model is in Chapters 5 and 6). Throughout the book, including the analytic history and descriptions (in Chapters 2,3, and 4) I portray open source as a social phenomenon, like any difficult collaborative project. It is also a political phenomenon because collaboration is 'governed' by formal and informal institutions, norms, and conflict-management procedures. And it is self-evidently an economic phenomenon as well, in both the micro and the macro sense. At the center of the process are individuals who engage in some kind of cost-benefit analyses according to some kind of utility function. And open source has real implications for the organization of production, for corporate structures, and possibly for the economy as a whole.

[15] Frederick P. Brooks, The Mythical Man-Month: Essays on Software Engineering. Reading MA: Addison Wesley, 1975.

All models simplify reality, and all analytic perspectives have baselines, be they implicit or explicit. My goal here is to take the political economy perspective seriously, but not too seriously, as a baseline. It would be taking it too seriously to posit that the lack of money is *the big puzzle* to be explained (although it is certainly part of the puzzle). It would be taking it too seriously to doubt or ignore what are obvious truths about people: that human beings often have a passionate relationship to their creative endeavors and their work; that they wish to share their creativity with others; and that 'value' inheres in things other than monetizable rewards. Each of these attitudes exists within the open source community. But none of them are unique to that community or distinctive to the 'information age'.[16] Human motivation and behavior is always and everywhere an elaborate mix of factors. An analytic perspective grounded in economic assumptions is a useful heuristic that helps to start structuring a story that explains that behavior. But a starting point is all it is.

The Bigger Picture

The other purpose of a heuristic is to open up discussion toward much broader questions that surround and embed the open source process. I will do that throughout this book in at least four general areas.

The first is simply the context of the Internet 'revolution'. The collapse of the dot-com stock market extravaganza can lead to unadulterated pessimism that is just as intellectually diverting as was the 'irrational exuberance' of the 1990's boom. Open source too has ridden some of the waves of public interest and hype over the last few years, with particular attention focused on Linux. Recognize right now that the future of the open source process is a bigger question than the future of Linux. Linux will not last forever. Someone will break Linux up and use pieces of it as a tool kit to build another

[16] In The Hacker Ethic and the Spirit of the Information Age (New York: Random House, 2001) Pekka Himanen explores these three characteristics as central to the "hacker ethic". I agree with this portrayal but I don't share the notion that these are somehow distinctive to information age work; in my view they have characterized human motivation for a very long time (think of cave paintings).

major operating system or something else. Remember at that point what is potentially durable and possibly deserving of the term revolutionary – not a particular manifestation of a process but the process itself. After all, the logistics 'revolution' was not any single container ship or a company building tractor-trailer trucks, it was a new way of moving goods around the world.

The rapid introduction into human affairs of extensive telecommunications bandwidth, configured as a neutral network, does not change everything.[17] But it does change some very important things about the constraints and opportunities that face individuals and organizations in the 'new economy'. The open source story opens up a significant set of questions about the economics and sociology of network organization, not just network economics. And it demonstrates the viability of a massively distributed innovation system that stretches the boundaries of conventional notions about limits to the division of labor.[18]

There is a subtle but important point that will emerge here, overlapping with Lessig's case that in a computational environment software code plays a structuring role much like law does in conventional social space. The open source process is a bet on the idea that just as important as the code itself and probably more fundamental is the *process* by which the code is built. Consider a slightly different analogy, to physical architecture not law (after all, if Lessig had been an architect not a lawyer he would have likely emphasized physical structures and his book might have titled "Code and other *buildings* in Cyberspace). Stewart Brand wrote that ''all buildings are predictions. And all predictions are wrong." His point is that some buildings are designed to 'learn' from their users over time and others are not; and that matters much more in the long run than what the building looks like on the day it opens.[19] Human-computer interface designers are deeply aware of the fact that what they build embodies decisions about policy, and underlying that rights, values, and even basic philosophical views on human action in the world.[20] But they have paid less attention to the process by which those

[17] Chapter 8 discusses in detail the notion of a neutral network and its implications.
[18] Gary S. Becker and Kevin M. Murphy, "The Division of Labor, Coordination Costs, and Knowledge," The Quarterly Journal of Economics November 1992, pp. 1137-1160.
[19] Stewart Brand, How Buildings Learn: What Happens After They're Built, New York: Viking Press, 1994
[20] A good review is Paul Dourish, Where The Action Is: The Foundations of Embodied Interaction, Cambridge, MA: MIT Press, 2001

decisions about design are made. Open source is one sign that the information politics discussion is growing up and taking itself seriously enough to confront those tricky questions. Some of these questions and their evolving answers will have significant and long lasting consequences beyond the lifespan of Linux or any other particular open source software program.

The second broad area is the evolving relationship between communities, culture, and commerce. The open source 'community' (as it calls itself) is indeed marking out a set of organizing principles. These include criteria for entering (and leaving), leadership roles, power relations, distributional issues, education and socialization paths, and all the other characteristics that describe a nascent culture and community structure. At the same time the community is figuring out how it relates to commerce and the capitalist economy that embeds it. These characteristics are evolving and are not always transparent. And the technology that lies at the heart of the community sometimes distracts attention from what may become really important changes in the way people relate to each other around creativity and economic production.

Peter Drucker argues consistently that technology may change the costs of doing things but that is ultimately a marginal adjustment in political-economic behavior. What make a significant difference in human life are the ideas, theories, and institutions that are themselves a product of experimentation and imagination, of a different sort. The steam engine was the metal behind the first industrial revolution, but the revolution was a set of ideas about organizing factories, limited liability corporations, trade unions, and daily newspapers. The second industrial revolution was a story about the publicly traded corporation, the commercial bank, business schools, the professionalization of women, etc. None of these are technologies, and neither is the open source process. They are each ideas, ideas that create institutions and ways of organizing that were nearly unimaginable beforehand and nearly unrecognizable when they first emerged.

My point is that during the early stages of economic and social change analysts often pay more attention to what is going away than what is struggling to be born. To use Schumpeter's phrasing, it is easier to see precisely the destructive side of creative

destruction, than it is to see the creative side.[21] We know how the old and familiar institutions function and we know when they are being challenged. The significance and meaning of a new way of doing things is unfamiliar. It may not be a functional replacement for institutions that are being destroyed.[22] And there is always a great deal of noise to accompany any signal. That counsels caution, but it also recommends an open attitude toward unfamiliar possibilities that demonstrate themselves even within relatively specific economic and social conditions.

The third general area is the nature of collaboration and production in knowledge-intensive economic processes. The software world is almost a limiting case for the study of knowledge economies, in the sense that it is made up of digitally-encoded knowledge all the way through from top to bottom.[23] Production processes that evolve in this space are not a hard test of limits but rather a leading indicator of change and a place where experiments can be seen at a relatively early stage.

Open source is an experiment in social organization for production around a distinctive notion of property rights. The narrow problem in property rights is simply who owns what. While this remains important, the open source process is playing around with more fundamental aspects of property. That is, what does it mean to 'own' something? What rights does 'ownership' confer upon whom and for what purpose?

The intuition around 'real' property is that to own something is to be able to exclude non-owners from it. The right of exclusion is essential because it brings with it opportunities to sell access or transfer the right of exclusion to someone else, under terms that the owner can set. 'Free-riding' is an unfortunate imperfection that governance systems try to minimize. For intellectual property, copyright and particularly the fair use provision is a pragmatic compromise between the interest of the owner-creator in having exclusive rights and the aggregate interests of society in gaining access to ideas. All of this sounds intuitive, but none of it is encoded in the facts of nature.

---

[21] Joseph Schumpeter, Capitalism, Socialism, and Democracy, New York: Harper & Row, 1942
[22] John Ruggie "`Finding Our Feet' in Territoriality: Problematizing Modernity in International Relations," International Organization, 47, 1, Winter, 1992, pp. 139-75
[23] Francois Bar, "The Construction of Marketplace Architecture," in The BRIE-IGCC E-conomy Project, Tracking a Transformation: E-Commerce and the Terms of Competition in Industries, Washington, DC: The Brookings Institution Press, 2001

Open source inverts the idea of exclusion as a basis of property rights. *Property in open source is configured fundamentally around the right to distribute, not the right to exclude*. It is almost as if fair use was extended without boundaries, along with a guarantee that no individual's fair use will be permitted to constrain subsequent fair use by any other individual, and for any purpose. What does 'ownership' then mean, and what is the significance of free-riding in this context? Ultimately the open source process poses a simple but provocative challenge: is it possible to build a working economic system around the notion of property rights as distribution? What kinds of characteristics would that system take on?

The fourth broad area is probably the most obvious. How 'big' a phenomenon is this, how broad is its scope? I will argue in this book for the demonstrated importance of the minimum case. Even if open source was just a story about software it would still be an interesting problem for social scientists thinking about large scale cooperation. And it would still have significant implications for economic growth and development.

At the same time I will build the contours of a more ambitious case, that the open source process has generalizable characteristics, that it is a generic production process, and that it can and will spread to other kinds of production. The question becomes, are there knowledge domains that are structured similarly to the software 'problem'? If we take the structure of the knowledge domain and the nature of demand that people have to solve particular kinds of problems in that domain as independent variables, then allow organization in the broadest sense (how people organize themselves to solve the problem) to be the dependent variable, can we sketch out some of the boundaries within which an open source type process might work? In addition to the practical implications, this is a reasonably good test of how well we really understand what makes the open source process succeed.

The Plan of the Book

This introductory chapter tried to set the overall problem of open source in a political context and suggest some of the directions in which the arguments of the book will proceed.

Chapter 2 traces the early history of the open source process. It focuses on the interaction between technology, culture, and politics. It explains how open source grew out of early programming needs, how it established a technical aesthetic and a nascent political culture, and how both were affected by Justice Department anti-trust actions, corporate strategies, and changes in the way mainstream technology users behaved.

Chapter 3 gives a more precise description of the phenomenon. How does the open source process function? I present data of different sorts that capture an important piece of what we know about the open source process and the people who contribute to it. I use this to build an ideal-type description of open source as a production process for complex knowledge goods.

Chapter 4 continues the history of open source to the present. It was in the 1990s that open source acquired a self-conscious identity as a community. By the end of the 1990s open source had become a phenomenon, in product markets, capital markets, mainstream corporate settings, and in the imagination of many people inside and outside the software engineering world. This history demonstrates the viability of (at least) two discrete ways of organizing the production of software, each of which is embedded in distinctive notions of property, policy, business models, and ideas in the popular imagination.

Chapters 5 and 6 are the core explanation of the open source process. I break the explanation into two separate buckets along the lines I suggested in this introductory chapter. The first is about microfoundations. Why do individuals choose voluntarily to allocate a portion of their time, effort, and mind-space to the open source process? What is the economic logic of the collective good that is being built, that frames these decisions? This is the focus of chapter 5. The second bucket is macro-organizational. How do individuals work with each other and govern that process, in a way that makes their contributions add up to a joint product? This is the focus of chapter 6.

Chapter 7 explores some business and legal issues. The open source process does not exist on its own; it enters an ecology of business and law that is densely populated by institutions that do things according to a different set of principles and rules. Yet open

source needs to interact, deeply and effectively, with existing structures of capitalist economies and legal structures like copyright, patent law, and licensing. The most interesting set of propositions begin to emerge where the two interface.

Chapter 8 summarizes the book and draws out implications. I explore some nascent answers to the broader questions that open source raises about political economy in the Internet-enabled environment. And I spell out implications for thinking about the problem of cooperation in a general sense, and specifically in an international setting. The general topics are: the importance of shifting property rights, the function(s) of a technological commons, the logic of distributed innovation, technology transfer and economic growth, new forms of power within networked communities, and the very interesting and timely problem of understanding interactions between differently structured organizations, specifically networks and hierarchies.