The Economics of Open Source Software

A Public Comment Submitted to the
Federal Trade Commission and Department of Justice Hearings on
Competition and Intellectual Property Law and Policy in the
Knowledge-Based Economy

Zoe Konovalov

November 4, 2002

Table of Contents

Introduction

The world of open source software and open standards was once seemingly the concern only of computer specialists and hackers, irrelevant to the general public or to economists. Yet in the past few years, public events have launched open-source software into the public eye. During its antitrust trial, Microsoft's defense argued that the company had about a 90% share of the operating system market, it was not able to abuse monopoly power, since viable alternatives existed which the market could turn to. Richard Schmalensee, Microsoft's economist, testified that Linux, an open-source operating system, represented a rapidly growing threat to Windows.[1] Companies like RedHat, a Linux distributor, and other open-source related firms were included in the technology stock mania of the late 1990s; RedHat's stock, for example, reached a high of $150 per share in the third quarter of 2000 before plummeting to around $6 a share today.[2] In 1998, Netscape announced that it would release the source code for its internet browser in an attempt to regain its position against the Microsoft juggernaut. They were influenced in large part by the writings of Eric Raymond, an open-source philosopher and advocate.[3] And in markets such as web servers, open source solutions like the Apache group have actually captured a dominant market share.[4]

Many people found it hard to believe that software available for free, and often cobbled together by the efforts of unpaid enthusiasts, could actually challenge Microsoft Windows or any closed-source software company on its own turf. Yet, in fact, the form of software development referred to as "open," in which the source code for the program is

---

[1] US Department of Justice, Microsoft appeals brief, 1999.
[2] RedHat, 10-K Annual Report, April 2001.
[3] Quitter, *Speeding the Net*, p316.

available, and typically based upon the contributions of a core group of volunteers, with extra help from casual amateurs, has consistently succeeded in producing programs which ival and even surpass their proprietary counterparts.

Press coverage of the distinctive culture of hackers – their inflammatory diatribes against "evil Microsoft," their libertarian leaning and fierce meritocracy – has been the new, trendy subject in the high tech field. Yet there has been very little economic analysis of why this strange concept actually works. How can the work of volunteers, who are paid nothing and give away their code, rival the product of a company that pays people to work? And why would it ever be in a company's best interests to give away its "crown jewels" – its source code?

In this comment, I will provide a definition of open source software and analyze various types of open source development, along with a brief history of open source software for context. I will discuss some economic analyses of different business models incorporating some form of open source software, mostly focussing on the labour side (i.e., what are the incentives programmers face to contribute). Then I will discuss the various business models that have been proposed for making money from open source software.

Definition of open source software

A computer program is a series of instructions of tasks to performs given to a piece of hardware. There are a number of different "languages," roughly analagous to human languages, which can be used to write a computer program. However, the only instructions which hardware can understand are a series of 1's and 0's, representing the presence or absence of an electrical current in a circuit. Computer programming languages – for example, C,

---

[4] Netcraft annual web server survey, www.netcraft.com

C++, Java, BASIC, or Perl, exist to provide an interface between the "machine language" of 1's and 0's, which is extremely tedious for the human mind to work with, and human thought about the goals of a program. Specialized programs, known as "compilers," can translate programs written in a higher-level programming language into machine language.

```
#include <stdio.h>
main()
{
    printf("hello world");
}
```

--The ubiquitous introductory computer program, written in C, this causes the words "hello world" to appear on the screen. (Fig 1)

Therefore, when a company or individual distributes a computer program, they can make a choice about whether to distribute only the compiled code, which appears as 1's and 0's and from which it is virtually impossible to recapture the original program, or they can distribute the higher-level source code, which carries much more information about the structure and purpose of the program. The most basic definition of open-source software is a software program for which the source code is available. The price of the software is irrelevant to this definition, although it is often the case, because of the culture of computer programmers and some other aspects of the software world, that open-source software is available for free over the internet. However, closed-source software can also be distributed for free; then it is known as "shareware" and usually used to build demand for a later or more sophisticated version. And open-source software is not necessarily free; its copyright license can put a number of specifications on its use, including payment. The theoretical differences between open and closed-source software come from their different developmental and distributional models.

Open source software is a particular kind of public good, characterized by non-excludability (once software has been put onto the internet, it can be distributed quickly and freely) and perfect jointness of supply (one person's use of the resource does not deplete it).[5]

A brief history of software programming and open sources

Computer science is a relatively new discipline, and it was born in government and university programs that supported an open culture of sharing code and collaborating. Therefore, the idea of "open source software" is as old as computer programming itself, and can be compared to the idea of peer review of academic articles. The history of computer science, and the interaction of the public and private sectors, can be seen as the story of investors trying to balance the tremendous growth that sprung from encouraging open collaboration, while still trying to capture the value that resulted.

The "ARPANET", the prototype of the modern internet, was built by the Defense Department in 1969 and linked hundreds of universities, defense contractors and research laboratories.[6] It gave an additional boost to the sharing of code by computer scientists all across the country, and it wasn't long before early versions sprung up of the standard bulletin boards and email lists used by open source developers today. Some important centers of computer science research included MIT's Artificial Intelligence Lab, Stanford University's Artificial Intelligence Laboratory (SAIL) and Carnegie-Mellon University's computer science department. ARPANET helped create a critical mass of information by allowing an easy exchange of ideas between people from all these different places.

Another important hub, the XEROX Corporation's PARC (Palo Alto Research Center), provides an early example of the potential danger of over investment in open source

[5] In fact, as will later be argued, use of open source software can, instead of depleting it, actually increase its value for others in the community.

development. PARC created a huge number of important inventions, including modern mice, windows, and icon software interface, the laser printer, the local area network, and early prototypes of personal computers. Yet although the existence of PARC certainly contributed far more value to the world than it cost – it was a classic public good – XEROX failed to recognize the value of the innovations that came from PARC, and didn't capitalize on many of them. Eric Raymond, a leader in the open source community, describes PARC as place that "developed brilliant ideas for everyone else."[7]

The University of Berkeley and the development of UNIX

UNIX, the first advanced and important computer operating system, arose as a result of a characteristic collaboration between the semi-public (Berkeley's computer science department) and private (Bell Labs.) Ken Thompson, a Bell employee, modified the Multics time-sharing operating system, and realized that by writing an operating system in the C programming language instead of high-level machine language, it could be much more portable, flexible, and understandable. The first implementation of UNIX came in 1969 – the year that ARPANET was created – and after a decade of work UNIX had been successfully "ported" to several different hardware systems. As Eric Raymond says, "If Unix could present the same face, the same capabilities, on machines of many different types, it could serve as a common software environment for all of them. No longer would users have to pay for complete new designs of software every time a machine went obsolete. Hackers could carry around software toolkits between different machines, rather than having to re-invent the equivalents of fire and the wheel every time."

The history of software development can be usefully viewed in terms of the framework of the clash between a collegial, open model and a corporate, closed model. This

---

[6] Levy, Stephen, *Hackers*, p52.

clash is exemplified by the problems that sprung up in the relationship between AT&T and Berkeley. Their collaboration started breaking down when a group of computer scientists at Berkeley developed a distribution of UNIX called the BSD (Berkeley Software Distribution). However, their code used part of the source code that had originated from AT&T, and the company sued Berkeley to prevent them from offering a free competition to its operating system, which earned such lucrative licensing fees.

Meanwhile, in the late 1980s, the personal computer (PC) market was beginning to heat up, and a company called Microsoft was, through a collaboration with IBM, able to seize a dominant share of the operating system market for these domestically used machines. The increase in PC use and the explosion of the use of the Internet by private consumers was joined by a flowering of other companies that kept their code private and earned a profit, such as Netscape, the creator of the most widely used initial browser for surfing the internet. This essay will later discuss why the market of personal users with low computer skills and desire for applications with low networking benefits is particularly suited for closed-source development.

However, open source development never stopped during this time. Because the Berkeley BSD distribution was tied up in the lawsuit by AT&T, there rose a huge demand for an open-source operating system of an equal power and sophistication to UNIX (DOS and Windows were not respected by serious hackers). One of the most important projects was Linux, started by Linus Torvaldis, who wrote a kernel (based in large part on earlier work done by Richard Stallman of the Free Software Foundation).

---

[7] Raymond, Eric, "A Brief History of Hackerdom."

The birth of Linux

As befits any institution with a passionate following, the story of Linux's creation has become hacker legend. It started with a Finnish graduate student, Linus Torvaldis, who became frustrated with the UNIX operating system in 1991, and revamped it into the Linux core program. However, to say that Linus "created" Linux would be to misrepresent his most important accomplishment, which was the perfection of the open-source system of software development. As Andrew Leonard puts it in his book-in-progress on the history of open-source software, "Torvaldis exploited the Net's facility for bringing people together . . . using e-mail and Usenet, he nurtured a worldwide community of freely collaborating programmers."[8] This network of programmers, all of whom used Linux, suggested bugs that needed to be fixed, and submitted code that fixed those bugs. Programmers had used the open-source system ever since the first days of the Internet, but Linux is the biggest, most complex, and arguably the most successful open-source project ever to exist.

of Linux: a cute icon has become em of the Linux ity.

Linus continued to work programming Linux, but he also acted as a "benevolent dictator": he received submissions from anyone else using the operating system, chose the best ones, and periodically (sometimes even daily) issued new releases of Linux's source code, crediting the contributors. By 1994, Linux version 1.0 was ready for general distribution. The operating system was steadily improved, becoming more stable and more complete, with every new version numbered in such a way that potential users could make a choice either to run the last version designated "stable" or risk bugs to get new features.

The Linux core handles all the basic functions of an operating system, and is still being managed by Linus. Meanwhile, there is an expanding galaxy of Linux-compatible

applications, all of them "copylefted," which are available from different places on the Internet. Linux companies such as Red Hat and VA Linux add value by providing complete Linux packages, available on CD-ROM, with the Linux kernel and a coherent selection of accessories, some of which may be developed by the Linux company itself, others of which are simply reproduced according to their "copylefts." Last year there was a period of intense excitement in the investment community about Linux. Red Hat went public and at one point had shares valued at $150, creating a market capitalization of $8 billion for the company; however, that euphoria has subsided – Red Hat now trades at a more reasonable $26 a share.

Today the five-year-old operating system accounts for 25% of server operating systems and about 4% of desktop operating systems sold worldwide. That makes Linux No. 2 behind Microsoft in the server operating systems market and No. 3 in desktop software, just behind Apple. It's available for download from many sites, including www.linuxtoday.com, on CD-ROM packages from various companies such as Red Hat Linux, and even comes bundled with some Dell PCs. Despite dismissive public statements, a Microsoft internal memo[9] reads, "[Linux] poses a direct, short-term revenue and platform threat to Microsoft, particularly in server space . . . Linux is making a progressively more credible argument that OSS software is at least as robust -- if not more -- than commercial alternatives . . ."

---

[8] Leonard, Andrew, "The Free Software Project."

The growth of the Apache web server group

The Apache web server group is another open source project that started in 1995, when the most used server software on the web was a public domain HTTP daemon developed by Rob McCool at the National Center for Supercomputing Applications. A group of webmasters, who had developed their own private extensions and bug fixes, gathered together to join and coordinate these changes, something that helped them immensely with all of their individual jobs. The core group of developers, including Brian Behlendorf (who works for O'Reilly Software Associates), and Ken Coar (who works for IBM), formed the Apache Software Foundation, a non-profit corporation, in July 1999.[10] Their goal was to provide a legal framework for Apache's development, but members of the Apache community remain solidly pragmatic and focused on their regular jobs. The Apache web server group has a completely different culture and goals from the Linux group. Their motivation for collaborating in an open source environment is much more closely tied to the reduction of development costs in their existing work, while a contributor to Linux is more likely to be motivated by what Lerner and Tirole call "signaling incentives."

Today, web servers using the Apache software account for about 65% of the total

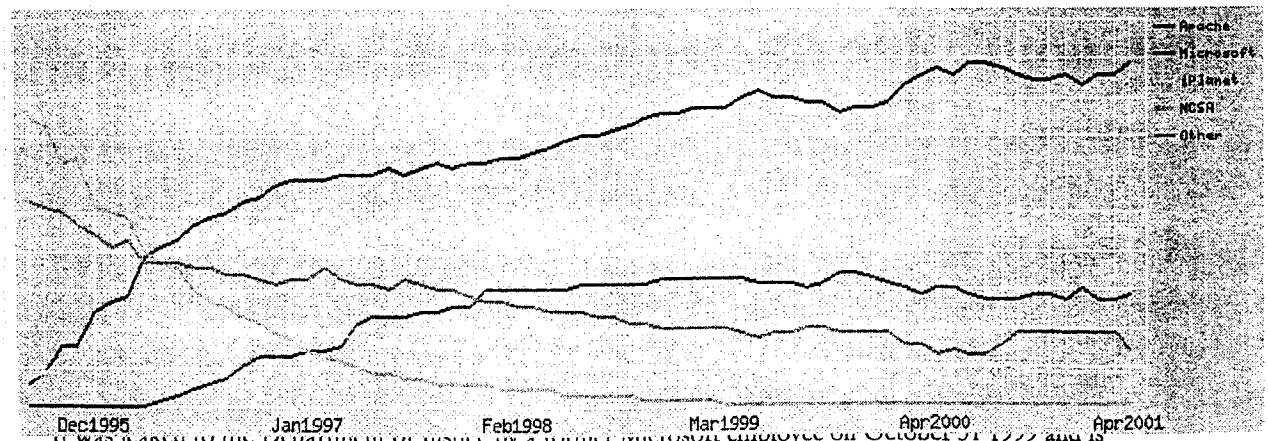

Fig 2 – Apache's growing market share (http://www.netcraft.com/survey)

It was leaked to the Department of Justice by a former Microsoft employee on October 31 1999 and is therefore referred to as the "Halloween Document." It's available in many places including on www.

[10] ww

market.[11] Every eight seconds another Apache-based website joins the existing 3.5 million on the Web.[12] Apache web servers are well known for being stable and secure; according to another survey by Attrition.org, over half of defacements of web pages were on Windows-based machines while Apache servers only accounted for 29% of defacements, despite a market share more than double that.

Netscape – Rise, fall, and possible recovery?

Netscape, which started as one of the early, trend setting closed-source software companies, decided in 1999 to go open-source. Why did they change their minds? It was originally very economically advantageous for Netscape to keep control of their source code. They had a "killer app," code which they had developed which did not exist anywhere else, and the market for talented programmers familiar with the then-nascent internet was illiquid enough that it would be extremely hard for other companies to duplicate their efforts. Moreover, the most important target users of Netscape were relatively computer-inexperienced users both in businesses and households. This meant that there was a fairly large training cost involved in beginning to use the Netscape browser. A company that trained all its employees to use Netscape would be unlikely to shell out even more money to train them again to use a different browser, and consumers would similarly be reluctant to learn again. Therefore, there was a large amount of profit for Netscape to capture by keeping control of its source code. However, over the course of the 1990s, Microsoft developed a competing Internet browser, Internet Explorer (IE), and began to use their monopolistic distribution channels to force IE's dominance over Netscape. By the late 1990s Netscape had lost a considerable amount of market share, and Marc Andreeson, the CEO of Netscape, citing the influential essay "The Cathedral and the Bazaar," decided to

[11] Netcraft web server survey, www.netcraft.com

release Netscape's source code and attract a group of open source programmers in an attempt to find a new way to challenge Microsoft.[13] So far, this attempt has been a relative failure, especially compared with the success of Linux and the Apache server, in attracting both developers and users. The Mozilla browser is an interesting case study to contrast to the success of Linux and especially Apache, to illuminate the market conditions and developmental choices that lead to incentives for programmers to volunteer their time to an open source project, and for consumers to prefer an open source product.

Hardware companies and open-source software: a rekindled romance

Since software and hardware are complementary goods, as will be discussed later, hardware companies have a natural interest in open source software. Indeed, IBM's mainframes originally all ran open-source operating systems, which IBM would collaborate on with its clients. However, during the 1980s IBM moved away from this model, closing the source code of its operating systems for mainframes and building a partnership with Microsoft to create the proprietary DOS (Disk Operating System) for its PCs.

In the past three years, hardware companies have been rekindling their romance with open source software. This is due in large part to the controversy over Microsoft's operating system monopoly, which it used to help push hardware towards the realm of a commodity. In May 1998, an IBM project team created a device driver that would allow the Linux operating system to run on its top-of-the-line mainframe computer, the System 390. IBM has announced investments in Red Hat Linux and has funded various Linux conferences. Other hardware companies are getting in on the action: Dell Computers now offers the option of having Linux bundled with its PCs, and Sun Microsystems has made significant investments in Linux server development.

---

[12] Netcraft survey, www.netcraft.com/survey

Open source volunteers and labor incentives.

Many participants in open source software development would probably deny that their behavior has anything to do with traditional economic incentives. Instead, they are more likely to cite the inherent ethical imperative to release source, or the fact that it is more fun and coding skills are improved better by working with other people. Yet during the 1990s – perhaps because of the increasing rise of the competing closed-source development model – the open source community began to be more self-reflective about the reasons for their success. If open source development could be viewed in light of the tragedy of the commons, where free-riders have an incentive to contribute less and take more than their share, might not the open-source world be inherently unstable and ultimately doomed to collapse? A number of important thinkers began to articulate the reasons why the tragedy of the commons was not the right metaphor and that, in some cases, open source development might not just be more enjoyable for programmers, but more effective.

Eric Raymond, a notable open-source programmer who has led such projects as the development of Sendmail, wrote an essay called "The Cathedral and the Bazaar," first presented in 1997. It was the first time anyone had put closed and open development models into an analytical framework that stressed the different incentives each model created and their effects on a project. The title or the essay comes from an extended metaphor he develops, comparing closed-source development to building a cathedral and open-source development to creating a bazaar. Before Raymond became familiar with Linux, he said, "I believed that the most important software (operating systems and really large tools like the Emacs programming editor) needed to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released

---

[13] Madden, Andrew, "The Good Ship Netscape," in *Red Herring*.

before its time." Instead, Linus Torvaldis, the creator of Linux, followed the rule "release early, release often, delegate everything you can, be open to the point of promiscuity . . . a great babbling bazaar of differing agendas and approaches . . . out of which a coherent and stable system could seemingly emerge only by a succession of miracles."

Traditional economic theory regards labor as an input like any other, which can be described by a characteristic production function. Therefore, according to traditional managerial thinking, increasing the number of people on a programming team could reduce the completion time required by a predictable percentage. Or, each hour spent programming has a regular and predictable productivity. And it is conventional wisdom in the programming world that past a certain point, the productivity of each additional programmer on a project declines precipitously, because of communication and coordination overhead.[14] In traditional production functions, there are three inputs: labor, land, and capital; inputs begin to experience declining marginal productivity because of constraints on one of the other inputs. If a restaurant wanted to add more sandwich grillers, they would eventually run out of space to stand around the sandwich grill. In the programming world, the equivalent of this space constraint is mental space in a project: there are only so many others you can work with on a particular programming problem. (See Graph 1).

However, open-source development takes advantage of modular, parallelized development. Raymond's metaphor of an open-source project as a bazaar is misleading, since all open-source projects depend upon having an organized leadership, either one person alone or a group of people with a governing arrangement. Certain programming tasks, such as debugging, don't even require that an open source contributor be aware of any

[14] Brooke, *The Mythical Man-Month,* p50.

other aspects of development, if properly managed. Furthermore, the infrastructure of open source development – mailing lists and websites – encourages a modular structure of software development, similar in spirit to object-oriented programming, which allows many more programmers to work on a project without getting in each others' way.

Different working arrangements and incentives can have radical and surprising impacts on labor productivity, often through the process of parallelizing workers' efforts. That is the key to open source development – it increases the productivity and quality of programming so dramatically that, despite essentially depending on monetarily unpaid labor, certain open source projects can nonetheless compete with the best that the closed-source world can accomplish. As Raymond puts it, "Given enough eyeballs, all bugs are shallow."[15]

Why is this true? The success of open source development can be broken into several parts. The first is the availability of labor: programmers are willing to donate their time to open source projects. The second is the effectiveness of the organizational model: despite such motivational tools as the threat of firing someone or the promise of a bonus, which conventional managers depend upon, and the far-flung nature of developmental teams (connected by only an email list) and the sporadic involvement of individuals, open source projects tend to be more efficient than closed source projects in creating stable, elegant code and fixing bugs. And the last part is the ability of open source projects to grow – often despite strong leadership from the top, projects such as Linux or Apache are being continually added to and extended as the software market changes. How do programmers know what to work on and which direction is the best way to go?

[15] Raymond, "The Cathedral and the Bazaar."

The first question, the motivations behind programmers who donate their time, is analyzed by Josh Lerner and Jean Tirole[16]. Essentially, a programmer will participate in a project if the benefits she derives are greater than the costs of participating. These costs and benefits have both an immediate and a delayed component. Immediate benefits can include monetary compensation, personal benefit from fixing a bug or customizing a program, and the immediate cost is the opportunity cost of a programmer's time, measured by the benefits of the other work she could be doing.

Delayed payoffs include two separate motivations, career concerns (job offers, venture capital access) and the benefits of having a good reputation among one's peers. Lerner and Tirole cite this second benefit as being tied to ego gratification, but a second part of having a good reputation is the advantage that other programmers will be more likely to help you with your projects since they respect you, lowering the cost to you of bug fixes or information gathering.

Therefore, the question facing a programmer deciding whether to participate in a project is:

(Current benefit (= Wage + Personal Benefit) + Future Benefit (= Career Advantage + Reputation Benefit) >? (Opportunity cost of time)

Analyzing the signaling incentive (which Lerner and Tirole use to group the future benefits of project participation) yields the prediction that open source projects will be more attractive if the performance is more visible, and if the participation is indicative of talent. In fact, these predictions hold true. Open source projects are religious about listing contributors' credits – one of the worst etiquette mistakes one could make would be to

[16] Lerner and Tirole, "The Simple Economics of Open Source."

delete a contributor's name from a project. In his essay "Homesteading the Noosphere," Eric Raymond predicts that projects will be more popular if they are anticipated to become big and famous; this is balanced against the relative importance of the contribution a programmer is able to make. He extends and complicates the kind of argument that Lerner and Tirole are making by comparing the customs governing the development of open source software to the Lockean theory of land tenure. It might or might not be better to make a small contribution to the Linux kernel than a big contribution to an upstart new project.

Creativity and Programming Labor Incentives

Like any creative work, the incentives for programming are far more complex, various, and fuzzy than the simple desire for profit. Raymond lists several aspects of open-source development that seem to provide strong incentives for motivation and attentiveness. The first is that open-source development provides an opportunity to "scratch your own itch"; that is, programmers are self-selected by their interest in a particular project. The work, therefore, tends to be fueled to a greater degree by personal pride, as well as the personal benefit that comes from fixing one's own problem. Raymond says, "too often software developers spend their days grinding away for pay at programs they neither need nor love." In a widget factory, it doesn't matter if the widget welders were personally fulfilled and satisfied by their work. Of course, it's always possible to make up stories about the worker who is resentful and daydreams, losing track of the widgets, or an increased number of sick days because people can't bear to drag themselves to work, but to a large degree, a widget is a widget and the inner life of its producer is irrelevant. But the mind frame of a programmer, or anyone doing creative work, is vital – motivation provides a much better source of sparks of inspiration than duty

Programmers' incentives and open source copyrights

The terms of the copyright used for an open source project has a profound influence on the course of the project and the incentives behind contributors, as well as the likelihood of "forking." There are several main types of copyright. The most frequently used is the GNU General Public License, or GPL, which was written by Richard Stallman in 1983 when he led a group of computer programmers in an effort to create an entirely open version of UNIX. The GPL is known as a "viral" license, or "copyleft," because it specifies that although anyone is free to take the source code and use it and customize it, any new code that incorporates some part of the original code must also be governed by the GPL. That is, it is impossible to take a program copyrighted with the GPL, modify it, and release it for commercial purposes. (Objections to the overly powerful viral nature of this license led to the creation of the GNU Lesser General Public License, which allowed users to incorporate certain free libraries into commercial products.) This contrasts with the BSD-style free license, in which users are free to do whatever they want with the code, including using it for commercial purposes, as long as they include an attribution to the copyright holder, and refrain from using Berkeley's endorsement without permission. Other academic institutions such as MIT, as well as more commercial open-source organizations such as the Apache Group, all use a version of the BSD license in which the only requirement is attribution. Netscape's Mozilla project did not use either of these licenses, but created the "Mozilla License," which some commentators pointed to as a source of tension with the open source community, since it reserved for Netscape the right to use donated code in proprietary ways.

The most important reason that some open source licenses would forbid the use of free code in proprietary ways is to preserve the incentives of programmers to contribute to

open source projects. If programmers are responding to the prospect of delayed payoffs, in the form of increasing their reputations, then closing the source of code that includes their contribution limits that signaling capacity. Furthermore, programmers understand themselves to be participating in a symmetrical gift culture, in which their contributions are freely given and are understood to result in contributions from others in the future. Modifying and closing open source code also increases the likelihood of "forking," or divergence of program development, which confuses users, splits programming resources, and is generally unhealthy for the life of a software project. On the other hand, forbidding the use of open source code for proprietary purposes can be very limiting, and could drive people who would otherwise be attracted to the community away. The Apache license by no means forbids the proprietary use of Apache code, since most members are attracted to the Apache group by the prospect of cutting development costs in their corporate webmaster jobs.

Recent work by behavioral economists such as Samuel Bowles and Herbert Gintis suggests another reason why it could be important that others are not allowed to use open-source material as part of a closed-source product: reciprocity. Experiments suggest that the idea of equality or fair play is valuable to humans even independent of the question of personal welfare. One experiment suggests, for example, that people willingly pay their fair share in taxes until they perceive that some businesses or individuals do not and escape consequences from the Internal Revenue Service.[17] Similarly, due to the culture of the open-source world, the programmers who participate in it value fairness in and of itself, another possible factor in the framing of open source licenses.

[17] Gintis, "Beyond *Homo economicus*: evidence from experimental economics"

Why are open source programs more efficient, elegant, and stable?

One of the tropes of the open source world is that open-source programs tend to be more "elegant" than closed source programs. It is hard to understand this concept without having programmed before, but a useful simile might be of a program as a complicated piece of machinery with places for other machinery to attach. Two machines might do exactly the same task equally well, but one machine could have an easy slot to attach another extension, while the second machine might need to be gutted and reassembled before it could be extended. Things are further complicated by the fact that many different programmers will work on a piece of code over its lifetime, and often, later generations of programmers will have no way of communicating with the original creators of the code, and will have to rely only on the code and its comments to understand how the program works and how to extend it. Style and commenting is an incredibly important subject for programmers, and it is common for a programmer to understand a piece of code she has written very well, while it is almost incomprehensible to others.[18] This is why there is a difference between writing a piece of code for personal use and for others: if the individual programmer is the only one who has to use, add to and understand the program, she can often complete the project in a fraction of the time it would take to create a stable, elegant, public-ready version.

For this reason, effort expended on comments, style, and overall organization of code can be seen as a public good. Comments and structure are often seen as a hassle by programmers, who prefer to concentrate on the immediate goal of getting code that works without bothering to explain themselves or figure out the best way of structuring their

[18] A contest exists called the "Obfuscated C Award," in which, as a joke, programmers compete to create C programs which are bug-free and work perfectly, but in which it is completely impossible for a normal human to figure out what the function or end result of the code will be after it is compiled. This can be seen as the extreme case of non-extendible code.

program. They impose an immediate cost; however, they drastically reduce the future cost of extending, modifying, and stabilizing a program.

One could say that there is no basic reason why open source programming should be able to create code of more elegance and stability; after all, managers in closed source companies could simply take account of the future savings and instruct their programmers to code in a future-savvy way. However, there are several reasons that programmers working on closed-source projects are much less likely to invest effort in the future value of their code. The first is a failure in managers' abilities to check up on programmers. Often, managers are inexperienced programmers themselves, and might not understand the implications of stylish coding. Even if they know how to code themselves, it takes a long time to read over the structure of a program, checking for grace and appropriate comments, as opposed to simply checking if it fulfills the end requirements. Managers sometimes try workarounds, such as programs that analyze a piece of code to calculate the percentage of the total text taken up by comments. However, programmers easily get around these ploys by inserting dummy comments such as /*Hello, World!*/. The more basic problem is that programmers themselves often have little investment in the future life of a project. They know that they will probably not be the ones working on a program later, so they will not have to deal with the future implications of their ugly code, and because it is not very transparent who is responsible for a piece of code, they have no reputation benefits to worry about.

Factors internalizing the incentives for programmers to create clean code	Factors destroying incentives for programmers to create clean code
Ownership is obvious	Tenuous link between code and its author
Code is an important platform for future projects	Code will not be used in the future
Link between cleanness of code and compensation: 1. Well-informed, insightful managers 2. Peer review (the larger and more widespread, the stronger the incentive)	No link between code quality and compensation
Programmer has a long-term relationship with code: expects to keep job for some time and to have to work with code created	Programmer expects to "hand-off" whatever code has been created and soon work at a different, unrelated job

Conventional software development and the principal-agent model

In short, there is a moral-hazard problem with the conventional model of software programming teams, which can be approximated by the principal-agent model in game theory[19]. Assume a project manager can choose a compensation package to offer to a programmer. The programmer can choose a high-effort working strategy (EH) including future planning and extensive commenting, or a low-effort strategy featuring murky, buggy code (EL). However, this model is complicated because the firm's initial profit will always be A; regardless of how clean the code is, if it is closed-source, it only matters that the code fulfills its desired function. However, the programmer's effort will influence the company's future profits, which depend upon being able to adapt, extend, and maintain its existing code. Low effort (EL) implies a higher probability of future profit B and effort EH implies a higher probability of future profit C, where (C>B).

Incentive schemes that are used to deal with the moral hazard problem include the pure wage scheme with a fixed salary, the pure franchise scheme where the agent pays a fixed franchise fee and is essentially a "residual owner," and the wage plus bonus scheme

where risks and profits are split between the agent and the employer. Game theory predicts that the franchise system and the bonus scheme will be more likely to elicit effort from the agent, respectively, and all of these payment schemes exist in the software industry.

However, the fact that the costs of inelegant software design are revealed only in the future means that the various incentive schemes are often irrelevant under a closed model, as long as the future is discounted heavily enough in people's minds, which it often is in the software industry. Decisions are made based on the initial profit A, not the future profits B or C.

An example of the problems that can arise from the shoddy workarounds encouraged by the organizational structure of closed source software companies comes from the example of Netscape.[20] In 1999 the company decided it wanted to release their browser code to the world and add a powerful new rendering engine. However, first it had to hire people to comb through the browser source code and remove all of the swearwords and other offensive language inserted in the comments by stressed programmers. Then, the structure of the Netscape code which had accumulated over the years turned out to be too complicated for outside programmers to understand well enough to modify. The Mozilla team ended up throwing out most of their old code altogether and starting completely from scratch.

By contrast, in the open source world, it is imperative that one's code be clearly understandable to others in a wide audience. That is how one's performance is measured and how one's reputation is increased. Credit for code is clearly assigned. Even within various departments of a company like Microsoft (for example, teams working on Microsoft Word and Internet Explorer), it is hard to request access to source code. By contrast, a

---

[19] Dutta, Prajit, *Strategies and Games.*

programmer releasing code into the open source universe knows that it must be readily comprehensible to a virtually unlimited audience. The transparency and peer review of the open source world seems to be the only effective mechanism for creating the public good of clean, elegant code. It is this fact, along with the idea of parallelizable debugging, that leads to the notorious fact that open source programs tend to be much more stable and bug-free than their closed-source counterparts. This feature, of course, is more valuable for mission-critical programs than for those used in a more casual way. A domestic Windows user doesn't mind that much, if she has to reboot her computer a few times when it freezes, but it is imperative that a web server not crash for months at a time.

Why are open source programs hard for neophytes to understand?

The vast majority of domestic consumers in the world still use application software created by proprietary, closed-source companies, and, furthermore, have no interest in seeing the source code of the software they are using. This fact is often hard for the hackers in the open source culture to understand; to a hacker, the idea of using a graceful program you can respect is paramount. However, there is a reason that proprietary software is much more accessible to inexperienced users: closed-source companies have the resources to expend on help files, documentation, and, most importantly, user interface architecture. (There is a distinction to be drawn here. All the help, documentation and guidance one could ever want on an open source program is certainly available online, and the programming community is quick to respond to requests posted on bulletin boards. However, this is more useful to the experienced or business user who knows where to go and what questions to ask, not Mr. First-Time PC user who needs a cheery pop-up help file).

[20] Quittner, Joshua, *Speeding the Net*, pp103-20.

Why doesn't documentation and help files get fulfilled by Raymond's free market of patches, where hackers try to "homestead the noosphere" in the most desirable places? There are several reasons. First, documentation and help is a bit of a dull job, which requires time but not much intellectual creativity; it therefore carries less signaling power to a hacker's skills. Second, a related point, programmers who volunteer their time are competing for status amongst each other, not amongst inexperienced computer users who have no relation to their lives, so they tailor their products to an experienced audience. And third, there is a higher information gap even assuming a hacker did have the motivation to make a program more usable. A fundamental feature of user interface architecture is that it is impossible to see the shortfalls in a system you have designed or that you understand. Yet the open source community doesn't have any of the UI testing labs that Microsoft, for example, invests in, and anyone who is exposed to an open source project in the development phase has already overcome a fairly high barrier of competency. Therefore, the cost to an open-source programmer of figuring out how to best help and guide an inexperienced user is higher than to a closed source programmer, and managers in closed-source companies can easily allocate programming time to the dull jobs of documentation and help, or the more complicated and inaccessible job (more related to psychology than programming) of user interface design. There is a reason that the closed-source Macintosh and then Windows were the first to use the famous "graphical interface," and that it is Microsoft and not Linus that has dabbled with such concepts as the "pop-up help paperclip."

It can therefore be predicted that if extensive help manuals and user interface design does exist relating to the open source world, it will not be free. In fact, this is the case, and the greatest number of companies providing help and packaging has sprung up around the program with the most potential to reach a domestic audience: Linux. (Anyone who is really

in the market for a web server is probably computer-savvy enough to be on a similar level with its developers and therefore understand the kind of help they are able to give.) The most prominent companies of this sort include Red Hat Linux and TurboLinux; essentially, what they sell is not the Linux code (which is available for free anywhere) but easy to use packaging, instructions for installation, and "dummy" bug fix guides. Part of the cost is from the raw materials of the box, paper and CD-ROM, but most is for the marketing and user interface design investment. The Red Hat user's manual, for example, has a glossy cover, and 425 pages of well-organized help sections, complete with page views and graphics and reassuring text like "If you count yourself among the many who are discovering Red Hat Linux for the first time, this book is for you!"[21] Nothing remotely similar would have been available for a would-be Linux newbie even three years ago.

4. Open Source Business Models and the Market for Lemons

In the previous section I gave the example of certain companies that have sprung up around the idea of providing user interface support for Linux, such as Red Hat Linux. Indeed, while the labor side of open source software is certainly most prominent, and has been the subject of most academic scrutiny, there are a number of business models that have been built around open source software in some way. The question of what to do with the source code of its projects is an important one for businesses, too often given little consideration under the knee-jerk reaction of keeping it secret. While the alternate response of many of the more fervent members of the open source community – that of freeing all source code – certainly doesn't always make economic sense for businesses, I plan to analyze

[21] Redhat, *Getting Started Guide*, pix

some of the various costs and benefits of releasing the source code for programs, and the dynamic incentives that govern how they change over time.

The case of companies contracting out a software development project – for example, a content management system, a database system, or a web server – is a classic example of asymmetric information and how it can sometimes create what Akerlof described as a "market for lemons."[22] Like a used-car salesman, a software company providing a package of closed-source code has a lot more information about the stability, elegance, and future maintenance costs of that software than the buyer ever will. Yet often the function of the code is far more vital to the buyer's business than a car ever could be.

	Good Quality	"Lemon"
Net valuation of buyer	1,500	750
Net valuation of seller	1,250	500

If what distinguishes a good car from a bad car is the expected costs of upkeep and repairs, similarly, the distinction between well- and poorly-written pieces of software is the future costs of changing and updating them. Suppose that the expected upgrade cost of a good quality software package is 100 and it is 850 for a badly-designed piece of code. If there are both competent and shoddy software consultants in the marketplace with an equal probability, and no way of evaluating the quality of the code inside the product they provide, the average valuation for a buyer will be (750 + 1500)/2 or 1,125. Yet no good-quality software provider will be able to sell at that price, and only "lemons" will remain in the marketplace.[23]

---

[22] Akerlof, "The Market for Lemons."
[23] Dutta, *Strategies and Games*, p392.

This is part of the reason why, in markets such as the business database system, brand names like Oracle become so important; they are one way of making sure of the quality of the code. Yet even a well-established company like Oracle has been infamous in the past for providing buggy first solutions to customers, as well as marketing strategies that play on uncertainties about their competitors. There are a growing number of mid-range companies in the business-database space who, while less prestigious than Oracle, have found their niche by offering open-source software solutions that solve the lemon problem through transparency.[24]

Another important advantages open source software holds for its customers is the security that they will be able to maintain and modernize their code even if the company supplying it goes out of business. In the closed-source world, there is a huge incentive for the selection of software suppliers with strong reputations and stable cash flows, because the risk for customers if their suppliers go out of business is huge. They will be stuck with an impenetrable system that nobody has the power to change when needed, and will be forced to incur the large costs needed to switch to an alternate system. In the words of Eric Raymond, "The price a consumer will pay is effectively capped by the *expected future value of vendor service* (where `service' is here construed broadly to include enhancements, upgrades, and follow-on projects)."[25] By contrast, if the source code for the programs a business uses is available, it doesn't matter whether the original supplier disappears, because the business can either make the necessary changes and adaptations itself, or hire somebody else to do it.

Opening the sources code also has a number of advantages for customers related to increased information. They will be better able to verify that the software system is stable and has no security bugs, by subjecting it to independent and outside scrutiny. It is a

[24] Scannell, Ed, "New front opens in database war," *Infoworld.*

common misconception of those unfamiliar with programming that it is necessary to keep source code hidden in order to prevent hackers breaking in. Raymond's words on the subject: "[If this is your belief], I recommend therapeutic conversation with a cryptographer immediately. The really professional paranoids know better than to trust the security of closed-source programs, because they've learned through hard experience not to. Security is an aspect of reliability; only algorithms and implementations that have been thoroughly peer-reviewed can possibly be trusted to be secure."[25] Even besides the obvious costs that come from outsiders compromising a system, the transparency that accompanies open source software makes it much easier for customers to check the quality of the code provided by their supplier.

Other benefits to the customer have already been touched on in this comment. Open source software is often cheaper than closed-source alternatives, or even free, depending on how much other demand there is in the marketplace for a particular type of solution. However, this is not always the case – for work contracted according to individual business specifications, there is no inherent reason why open-sourced software should be any cheaper. And there is also the benefit that comes from code of higher quality, because of the various labour incentives discussed earlier. Open source code is usually more stable and extendable than closed could ever be.

However, there are also aspects of open source software that can give clients pause. A particular problem comes from open source code that already exists as part of the public domain, not created according to specification. Open source programs are generally less user-friendly than closed source programs, because open source programmers are less willing to sink time into surface design, and other aspects of using the programs – understanding

[25] Raymond, "The Magic Cauldron."

documentation, finding help – are often much less accessible to non-techies than are provided by the more commercial closed-source software world. However, if one does have technical expertise, the service and support found in the open-source world of bulletin boards and email lists is usually at least as good as anything else available.

Finally, there is the marketing consideration. Among many parts of the corporate world today, open source software is dismissed out of hand as a business solution because it is assumed that anything offered for free, or as part of the open-source culture (seen as not "serious") cannot truly be of good quality. This image is often reinforced by open-source programmers, who, because of the culture of the world they operate in, often present an unpolished appearance and do not know how to interact by the rules of formal business. While this might not affect the quality of their code, it can make it hard for them to interact with some clients, depending on the nature of their corporate culture.

After examining these costs and benefits to the consumer, it is fairly easy to predict the type of customer for whom open source programs will be most valuable. This customer is searching for a business solution where stability, security, and quality has much more priority than surface design and usability features. The customer is technically sophisticated, so that finding access the support offered by the open source world carries less of a cost. Interestingly, the actual sticker price of a software program is comparatively unimportant to the customer in this profile; other, non-listed costs and benefits are much more important considerations. This makes it clear why, of all open source projects that exist today, the Apache Software Foundation has been the most successful. They specializing in developing mission critical software – web servers – on which their participants' jobs rely, and they have absolutely no need to make their work accessible to newcomers.

---

[26] Raymond, "The Magic Cauldron."

When does it make sense for a business to provide a program's source code?

It is important to distinguish between the idea of individual programmers making the cost-benefit decisions to contribute to a piece of open source code, and profit-driven businesses who make the decision to either release the source code of programs that they have initiated, or hire programmers to work on open-source projects. It would be theoretically possible for a flourishing open-source software universe independent of any related companies, thanks to the contributions from programmers who decide that volunteering time to open-source projects is a valuable enterprise. However, any phenomenon as successful and intriguing as open source software is certain to attract companies who seek some way of making a profit from it. So: how do businesses actually make money from open-source software? Isn't it a contradiction in terms? After all, open source software is usually free. Where does the money come from? We will see that there are a number of factors that could lead companies to release the source code of their "crown jewels". Companies will be more likely to do it with programs which are either specifically contracted, extremely unique and individualized, or with programs which are very generalized, but with a value that depends largely on network effects (programs which only become useful when many other people use them).

A company faces a trade-off when deciding whether or not to release the source code of its programs. It could possibly lose future revenue. If the software project has already been commissioned by a client, the issue is not the loss of that fee, but the future loss of other fees by clients who have similar problems, or the release of valuable information to potential competitors. On the other hand, open source development has a number of advantages. Development costs are lower and the code is more likely to be stable faster; it is likely that the customer could prefer it. Furthermore, open source code is a good

(Foreclosed future revenue) >? (Reduced development costs) + (Information effects) + (Network effects)

-Cost-benefit analysis for a company deciding whether to release source code for its program

signaling method when a market is competitive or crowded. If a company's software program is the only one of its kind to solve a particular problem, this is less important. But if there are many alternative options, a potential customer might not even be willing to consider a particular company if not for the proof of competence offered by its code. An example of this is the use of content-management systems by websites: the high-end option is the Oracle database, which is extremely expensive, closed-source, and uses extremely sophisticated technology. In the mid-range of the market, there are a number of competing alternatives for lower-priced content-management systems, none of which have market dominance. More and more of them are beginning to move to an open-source model of development. Since content-management software is not an extremely cutting edge technology, and furthermore, it is very specific, requiring much customer personalization, these companies do not lose much competitive ground by opening their source code, and make themselves much more attractive to their clients.

Netscape, for example, saw its revenue sources changed as its business developed and as its core browser technology became less-cutting edge. When companies such as, primarily, Microsoft, began offering competing browsers free, rent from Netscape's formerly proprietary technology disappeared. According to Netscape's 1999 10-K report, revenues from licenses of enterprise software and related professional services revenues accounted for 66.4%, in that year, up from 30.4% in 1996, respectively. The company was evolving into a service provider, which made the costs of releasing its source code much lower and the benefits much higher.

A final and very important incentive that pushes towards the opening of source code and standards is the extent to which a software program provides a base platform for other systems. At one end of this continuum would be something like the HTTP protocol that allows for computers all over the world to link into the global internet. In the middle of the continuum would be an operating system, which is installed on one particular computer but which provides a base for other applications to hook on to. And at the other end of the continuum would be applications, stand-alone utilities that hook on to an operating system and on which no other programs depend. Clearly, the value of network-crucial programs depends much more on their ubiquity and attractiveness as a base platform. The more that a program works as part of a system with other programs, the more pressure there will be for its standards to be open and its source code available, so that it can be useful as a platform. This is not so important for an application like a word processing program.

Network externalities in a traditional network come when a subscriber can reach others in a larger network. Although software programs are often governed to some degree by this network effect, they can be thought of a "virtual network,"[27] where network externalities arise because larger sales of component A induce larger availability of complementary components B1, ..., Bn, thereby increasing the value of component A. The increased value of component A results in further positive feedback. Despite the cycle of positive feedbacks, it is typically expected that the value of component A does not explode to infinity because the *additional* positive feedback is expected to decrease with increases in the size of the network.

Not only do applications gain less value from being open-sourced than networked programs, it is harder to make money from them. Raymond points to the distinction

---

[27] Economides, Nicholas, Information Networks website, http://www.stern.nyu.edu/networks/site.html

between the use value and the sale value of software (equivalent to its value as an intermediate and final good)[28]. Much more of applications' value is captured in the sale.

This prediction can be verified by real life results. All of the software that drives the Internet – the HTTP protocol, the Sendmail program which governs email standards, etc. – is open source. The open-source Apache web server software is the most popular choice for businesses; it is a mission-critical, networked program. In the world of personal computer operating systems, Microsoft Windows, a closed-source program, is dominant, but there is significant competition from an open-source program, Linux, among certain audiences. However, there is no open-source equivalent to Microsoft Word or WordPerfect, or even close.

Some business models of open-source software companies

After taking all these factors into consideration, we can determine a number of possible business models for companies who want to take advantage of open source software, and evaluate their potential success. The first person to look at open source business models in an analytical way was Eric Raymond, and his ideas were refined and expanded by Lerner and Tirole. In June 2000 Frank Hecker released a paper on the internet called "Setting Up Shop: The Business of Open Source Software," based on an internal memo he had written at Netscape in 1997 which spurred that company's decision to release the source code of its browser. These business models can roughly be divided into those that focus mainly on reducing costs of software as an input, and those that create profits from associated benefits of open source software.

1. The Apache model: cost-sharing. This business model relies on open source development not as a source of profits, but as a way to drastically reduce the costs for

[28] Raymond, "The Magic Cauldron."

companies, allowing their primary businesses to become more profitable. If different users all have a similar need for a complex type of software that is a primary input to their business, they can derive great benefits from sharing the cost of development and reaping the advantages of widespread peer review.

2. Risk-spreading. This is another form of cost-reduction, in which the encouragement of an open source community reduces the possible risk of a particular developer leaving or going out of business.

3. Market positioning. This is the use of open source development in one software market to entrench a position and allow for the sale of proprietary software in an associated market. One example is Netscape releasing the source code of its Mozilla browser in an attempt to keep Microsoft from gaining a browser monopoly, so that it could continue to hold a viable place in the server market.

4. Encourage the sale of complementary hardware. This was the earliest open source business model, since IBM originally released all the source code for the operating systems of its early mainframes. Hardware and software are complimentary goods; a fast chip is useless without software to make it work. Unlike software, hardware is a rivalrous good: if I own a computer, that precludes you from owning the same computer. It is therefore very simple to capture sale value from hardware, and open source software can be used as a way of increasing the value of that hardware. In recent months, hardware companies such as IBM, Hewlett-Packard, and Sun Microsystems have all announced various kinds of open source initiatives and alliances with the Linux community, so it is clear that this business model is an important and viable one.

5. Sell services associated with open source software. The old cliché would be "Give away the razor, sell the razor blades," although this is not completely precise since a

razor is a rivalrous good while open source software is not. A better analogy might be "Give away the recipe, open a restaurant." Companies following this business model include RedHat, which sells well-designed packages of ready-to-install versions of Linux, offers installation support and technical services.

According to RedHat's April 2001 10-K filing, it plans to make money by "offering technical support, custom development, and related services to customers, [offering] convenience and quality with shrink-wrapped packaging, user manuals and other related documentation and access to services and technical support offerings; using open source products as a means of attracting visitors to their web sites, which in turn can result in the sale of other products, services, and advertising; and developing brand loyalty . . . [to] leverage to sell additional products and services to customers."

6. Intellectual capital marketing. Prestigious consulting companies such as McKinsey and the Boston Consulting Group publish journals collecting their most important market research and business insights. It is true that this allows access to their work without paying a consulting fee, but their gifts are more than repaid by the marketing value of these journals, increasing the companies' intellectual reputation and their name recognition. In a similar way, a popular open source release, if associated with a particular company, acts like free advertising for the expertise of its human capital – its programmers – and can allow it to gain contracts for individual customization.

Conclusions

The success of open source software is not a fluke or an unstable cultural phenomenon. Instead, it is a serious and potentially revolutionary form of software development, which, although it seems counterintuitive to conventional business thinking, has the potential to challenge proprietary development in certain arenas. Some open source

projects have been extraordinarily successful. The Apache web server has a sterling reputation and a market share of 65% which continues to grow. The Linux operating system is an explosive cultural phenomenon which continues to become more user-friendly and technically sophisticated every day. Businesses are starting to take open-source ideas seriously – even if it's sometimes hard to adapt existing closed-source businesses to open-source frameworks. Netscape's experiment releasing the source of its web browser, as Mozilla, has had mixed results. The browser, which was made available for the widespread public in February, has some interesting technical advances, but was delayed more than a year in development and has not yet pulled the rug out from under Microsoft. Some of Netscape's problems related to their unusual choice of license; they also had problems building the kind of open-source development community that comes to a project started from scratch.

There are many open-source initiatives which may become important in the future. Apple's new Macintosh OS X, released in March this year, was based on an open-source UNIX variant; the company described it as "the most fundamental changes in both core technology and user interface design made by the company to the Mac OS in a single upgrade since the original introduction of the Macintosh in 1984."[29] Hewlett-Packard is developing an open-source project, "e-speak," which aims to provide the new backbone for internet trading in services[30]. Major hardware companies such as IBM, Hewlett-Packard, and Dell and working with the Linux community to develop new high-end servers. And open-source software has geographical as well as technical scope. Many developing countries, such as India and China, are beginning to look towards open-source software as a way to avoid the high prices charged by companies like Microsoft, and encourage a local software

[29] Apple Computers 10-K Annual Report, Sept. 2000, p11.

industry which can learn from the example provided by open-source programs. Part of the United Nation's new development program involves handing out CD-ROMs with versions of Linux to promote the use of open-source software.[31]

Yet the tools of traditional economics can seem frustrating when attempting to analyze the success of open source software development and business models. It is still hard to quantify the relationship between labour productivity, creativity, and inspiration, or the costs and benefits of network effects and asymmetrical information to companies. In this comment I have attempted to provide an overview of open-source software study, and a tentative exploration of the economic incentives underlying this surprising phenomenon. Yet the subject of the economics of open source software is still wide open, and the future will hopefully bring many more investigations into this important topic.

[30] www.e-speak-hp.com

[31] www.sdnp.undp.org/home/html, "Sustainable Development Networking Programme Description."

Bibliography

Books

Burdman, Jessica, *Collaborative Web Development: Strategies and Best Practices for Web Teams.* Addison-Wesley Longman Inc., New York, 1999.

Delamarter, Richard, *Big Blue: IBM's Use and Abuse of Power.* Dodd Mead & Co., New York, 1986.

Dutta, Prajit K., *Strategies and Games: Theory and Practice.* Massachusetts Institute of Technology Press, 1999.

Kernighan, Brian, and Dennis Ritchie, *The C Programming Language.* Prentice Hall, New Jersey, 1988.

Levy, Stephen, *Hackers: Heroes of the Computer Revolution.* New York: Dell, 1985.

Macho-Stadler, Ines, and David Perez-Castrillo, *An Introduction to the Economics of Information: Incentives and Contracts.* Oxford University Press, 1997.

Moore, Geoffrey, *Inside the Tornado: Marketing Strategies from Silicon Valley's Cutting Edge.* HarperCollins, New York, 1995.

RedHat Linux, *The Official Red Hat Linux Getting Started Guide*, Red Hat Inc., 2000.

Resnick, Mitchell, *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds.* Cambridge, Mass.: MIT Press, 1994.

Shapiro, Carl, and Hal Varian, *Information Rules: A Strategic Guide to the Network Economy.* Harvard Business School Press, 1999.

Quittner, Joshua, *Speeding the Net: The Inside Story of Netscape and How it Challenged Microsoft.* Atlantic Monthly Pr., New York, 1998.

Wayner, Peter, *Free for All: How Linux and the Free Software Movement Undercut the High-Tech Titans.* Harper Collins Publishers, New York, 2000.

Weinberg, Gerald, *The Psychology of Computer Programming.* Dorset House Publishing, New York, 1971.

Weiss, Mark Allen, *Data Structures and Algorithm Analysis in C.* Addison-Wesley Longman, Inc., Menlo Park, CA, 1997.

Journal Articles

Akerlof, "The Market for Lemons: Quality Uncertainty and the Market Mechanism," *Quarterly Journal of Economics*, 89: (3) 488-500, 1970.

Bowles S. and H. Gintis, "Walrasian economics in retrospect," in *Quarterly Journal of Economics* 115: (4) 1411-1439, Nov. 2000.

Gintis, H., "Beyond Homo economicus: evidence from experimental economics," in *Ecological Economics* 35: (3) 311-322, Dec. 2000.

Hannemyr, Gisele, "Technology and Pleasure: Considering Hacking Constructive," *First Monday*, vol. 4, no. 2 (Feb.), firstmonday.org/issues/issue4_2/gisle/index.html, 1999.

Kuwabara, Ko, "Linux: A Bazaar at the Edge of Chaos," http://firstmonday.org/issues/issue5_3/kuwabara/. (*First Monday* internet journal, Feb. 2000).

Lerner, Josh, and Jean Tirole, "The Simple Economics of Open Source," http://papers.nber.org/papers/W7600. NBER Working Paper, March 2000.

Newspaper and Magazine Articles

Ante, Spencer. "The Four Horsemen of the New Economy." *Business Week,* 10/02/2000, Issue 3701, p48, 1p, 1 graph, 1c.

Burr, Jeffrey; Lelii, Sonia. "The World According to Oracle." *eWeek*, 10/02/2000, Vol. 17, Issue 40, p1, 3p, 1 chart, 1c.

Cassidy, John, "The Force of an Idea," *New Yorker*, January 12, 1998.

Chang, Leslie, "Microsoft Bashers Bloom in China," *Wall Street Journal,* Jan. 1, 2000.

Lee, Lydia, "Linux in Every Lap," www.salon.com, Feb. 24, 2000.

Leonard, Andrew, "Apache's Free Software Warriors," Oct. 30, 1997. (http://www.salon.com/tech/feature/1997/10/30/feature/index.html)

MacVittie, Lori. "The Means to Communicate." *Network Computing,* 10/16/2000, Vol. 11 Issue 20, p112, 2p, 2c.

Madden, Andrew. "The Good Ship Netscape." *Red Herring Magazine*, September 1997.

McMilian, Robert, "IBM's Linux Point Man: Interview," Linux Magazine's Web Exclusive, http://www.linux-mag.com/online/iv_irvingwb_01.html, October 2000.

Ricadela, Aaron; Whiting, Rick. "Back to Data Basics." *InformationWeek,* 10/02/2000, Issue 806, p22, 4p, 3c

Scannell, Ed, and Sullivan, Tom. "New front opens in database war." *InfoWorld,* 10/23/2000, Vol. 22 Issue 43, p12, 2/3p, 1c.

Shankland, Steven, "Computing heavyweights to chaperone Linux into servers." *CNet News*, August 30, 2000. (http://news.cnet.com/news/0-1003-200-2648853.html)

Weiss, Aaron, "Open Source Moves to the Mainstream," *Information Week Online* (www.informationweek.com), April 20, 2000.

Websites

Apache Group, www.apache.org.

Attrition Security website, security survey, www.attrition.org.

Economides, Nicholas, Information Networks website, http://www.stern.nyu.edu/networks/site.html

Ghosh, Rishab Aiyer, "Cooking pot markets: a model for trade and services on the internet," http://www.dxm.org/tcok/cookingpot/.

Free Software Foundation, "What is Copyleft?", www.fsf.org/copyleft/copyleft.html.

Hecker, Frank, "Setting Up Shop: The Business of Open Source Software." http://www.hecker.org/writings/setting-up-shop.html.

Internal Microsoft memo ("Halloween Document") posted on www.opensource.org.

Leonard, Andrew, The Free Software Project, book-in-progress. http://www.salon.com/tech/fsp/outline/index.html, March 2000-April 2001.

Netcraft, Web server survey. http://www.netcraft.com/survey/.

Netscape Mozilla development site, www.mozilla.org.

Open Source Initiative web page, www.opensource.org.

Raymond, Eric, The Cathedral and the Bazaar, working paper. http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar, May 1997.

Raymond, Eric, Homesteading the Noosphere, working paper. http://www.tuxedo.org/~esr/writings/cathedral-bazaar/homesteading/, April 1998.

Raymond, Eric, The Magic Cauldron, working paper.
http://www.tuxedo.org/~esr/writings/magic-cauldron/magic-cauldron.html#toc9, June 1999.
Raymond, Eric, A Brief History of Hackerdom.
http://www.tuxedo.org/~esr/writings/cathedral-bazaar/hacker-history, February 1997.
Shirky, Clay, "In Praise of Evolvable Systems."
http://www.shirky.com/OpenSource/evolve.html
Sims, David, "Interview: Eric Raymond, Software Developer," *TechWeb,*
www.techweb.com/internet/profile/eraymond/interview, February 25, 1999.
Stallman, Richard, "The GNU Manifesto." 1984. http://www.gnu.org/gnu/manifesto.html
Stallman, Richard, "Why Software Should Not Have Owners." 1994.
http://www.gnu.org/philosophy/why-free.html
Torvalds, Linus, "Linux History," *Linux International,* at www.li.org/li/linuxhistory.shtml, 6 July 1999.
US Department of Justice, Microsoft appeals brief, 1999.
http://usdoj.gov/atr/cases/ms_index.htm

SEC Filings

Apple Computers Inc., 10-K Annual Report Form, filed September 30, 2000.
Microsoft Corporation, 10-K Annual Report Form, filed September 28, 2000.
Netscape Communications Corporation, 10-K Annual Report Form, filed January 29, 1999.
RedHat Inc., 10-K Annual Report Form, filed April 19, 2001.