Very preliminary draft: Please do not quote
without permission. Comments welcome.

The Economics of Open Source and Free Software*

Gerald P. Dwyer, Jr.
Federal Reserve Bank of Atlanta

Abstract

Open source and free software has been hailed as a substitute for proprietary software and greeted with great fanfare in a variety of forums (Raymond 1998.) It is possible to interpret such software as the work of hobbyists, since there is no monetary payment to programmers for writing such software. In this paper, I suggest that there is much more to open source and free software than hobbyists. In fact, open source software is interesting because its production can be interpreted as *collaborative development of software outside firms without monetary payments between the parties.* I will assume that all agents are motivated by their own well being. Why and under what circumstances will agents work on open source software? I provide a preliminary answer to these questions.

May 1999

* The views in this paper are those of the author and not necessarily those of the Federal Reserve Bank of Atlanta or the Board of Governors of the Federal Reserve System.

I. INTRODUCTION

Open source and free software has been hailed as a substitute for proprietary software and greeted with great fanfare in a variety of forums (Raymond 1998.) The most prominent recent example of open source software is the operating system Linux with GNU, though there are others. From an economist's standpoint, an interesting aspect of open source software is that people write software with no direct monetary payment for the software. By itself, this poses no interesting issues: some people do carpentry work on the weekend without direct monetary payment and that's not exactly earth shattering. If writing open source software is just a hobby, then the economics of it may be no more or less interesting than the economics of carpentry as a hobby.

In this paper, I suggest that there is much more to open source and free software than hobbyists. In fact, open source software is interesting because its production can be interpreted as *collaborative development of software outside firms without monetary payments between the parties.* I will assume that all agents are motivated by their own well being and yet will suggest that agents will write open source software under certain conditions. Why and under what circumstances? I provide a preliminary answer to these questions.

II. SOFTWARE

A. *Open Source and Free Software*

Open source software and *free software* are terms with specific meanings. As the terms are commonly used and are used in this paper, both open source software and free software are copyrighted and neither is in the public domain. *Open source* is a certification mark with a specific meaning (available at http://www.opensource.org/osd.html.) *Free software* in this paper means any copyrighted software licensed under the GNU General Public License (GPL) available at

http://www.gnu.org. This usage of the term *free software* has no necessary connection with software sold at zero pecuniary price. Free software could be sold for a positive pecuniary price, and software not licensed under the GPL is available for a pecuniary price of zero. Free software as so defined has been in use since 1984 and is an example of open source software.[1]

The names *open source* and *free software* come from two characteristics of the licences for such software. The first of these license provisions is a requirement that the software license may not restrict other parties from distributing the software and may not require a royalty for using or distributing the software. The second of these license provisions is a requirement that all distributions of the software include source code and that alterations to the source code be permitted. There are differences between the open source specification and the GPL, but these are less important for my purposes than these two common elements.[2] In this paper, I define the terms *open source software* and *free software* to mean any software with these copyright restrictions and I will use the two terms interchangeably.

Software distributed in source code form that can be altered by users is far from new (Ceruzzi 1998, Ch. 3.) While UNIX is not licensed under the GPL or any similar license, UNIX's development has some of these characteristics. Commonly, the source code is distributed and much of that source code is compiled for an individual machine. Users of UNIX had a great deal of involvement in the operating system's development (Salus 1995a.) Similarly, the development of

---

[1] In short, free software is a proper subset of open source software.

[2] The Open Source Definition interprets the GPL as consistent with its definition which indicates that the GPL is at least as restrictive as the Open Source Definition. The GPL requires that all software distributed with the GPL licensed software be under the GPL. Open source software can be combined with proprietary code and distributed subject to the terms of the alternative licenses.

the Internet included a great deal of involvement by selected users of the network (Salus 1995b, Bradner 1999.) Indeed, Richard Stallman (1999) began to work on GNU because software that earlier had been widely available in source code became available only under restrictive licenses with royalties. Source code to solve problems has been widely available on the Internet almost since its inception and still is available today, e.g. at Netlib (http://www.netlib.org.)

I do not mean to suggest that all of these examples are open source software, because they are not. Open source software explicitly grants permission to others to make changes to the source code and distribute the source code and any compiled versions with those changes.[3] Still, there is a substantial history behind open source and free software, which provides an illuminating background.

In the end, the perplexity seems to be this: people jointly write software that they then give away. You might think that I am trying to make the problem too hard: people simply write this software for a hobby and then give it away because they like to write software and it's not worth doing anything other than giving it away?

There is an example that is hard to square with such an easy solution. The Apache Web Server has been developed by people who run web sites and is used in their work. People who do this work regard it as a productive use of their work time for firms, not something that they do to relax after work (Behlendorf 1999.)

---

[3] There can be restrictions to maintain the integrity of the original author's source code.

B. *Background Propositions about Writing Software*

Some background aspects of computer programming are helpful for thinking about the conditions for writing software.

The first proposition is that software programs can be broken into pieces and that communication between the programs can be relatively simple.[4] The programs at issue are not monolithic entities that must be completely understood by one person or even a small group of people. Even an operating system (OS) such as Linux consists of many different programs. In fact, Linux itself is the kernel of the OS. Linux is useful only with many additional programs, e.g. drivers and *grep*. Drivers are separate pieces of software necessary for the kernel to function with any particular hardware. *Grep* is an example of the many programs that were written before the kernel of the OS was written. *Grep* is a utility that finds the files containing particular text, a much more sophisticated version of the *find* command in MS-DOS. Grep is a useful component of Linux, but grep was written before the source code for Linux and can be used in other OSs. One need not know very little about the OS and nothing about the computer hardware in order to write grep in C. Grep is similar to many other processes in the OS because the kernel calls these other processes to do work.

The second proposition is that there are serious diseconomies of scale in writing an individual program. This is evident in the casual references in the press to the small number of people who will work on a software team at any one time. Much of the literature on software engineering focuses on ways to reduce the coordination costs when there are more than a couple of programmers working

[4] If this proposition is unfamiliar, Salus (1995a) provides a discussion in the context of Unix.

on a program. The folklore suggests that coordination costs can become overwhelming with more than half a dozen programmers working on a single program.

In addition, there are two very different costs associated with any program. The first is the cost of writing the program; the second is distributing the program. The cost of writing a program is largely independent of the number of users.[5] The cost of distributing a program is an increasing function of the number of users and is approximately proportional to the number of users.

I will take it as given that software is continually being improved. Improvements can be adding functionality to the software and they can be correction of bugs.

Last, there are network effects associated with software programs. Anyone who has written a paper with another person knows about these costs. Writing a paper with another is much simpler if the joint authors use the same word processor. An OS is more useful if others use it because the availability of ancillary programs and information increases with the number of users.

III. Why Will People Write Free Software?

The question raised by the above definitions and citations to the literature is: Why would people collaboratively develop software outside firms without monetary payments between themselves or from others?

At first glance, it seems that the most extraordinary aspect of free software is programmers' inability to collect royalties, and it is hard to imagine that such a restriction is efficient. As it turns out, this is not so surprising. The more extraordinary thing is people's ability to organize themselves and write software with the only apparent coordinating device being their use of software.

---

[5] I say "largely" because the functionality of a program can change as the number of users increases if users are not identical.

A. *A World without Transactions Costs*

Imagine what software development would look like with zero transactions costs. As in Barzel (1997, p. 4), transactions costs here include all of the costs of buying or selling a good or service including the costs of acquiring information about the good. With zero transactions costs, everyone would know the marginal product of pieces of software, routines in the software, even individual lines. This does not imply that the world is deterministic. Stochastic elements could be present in the economy, but the value of the resulting software in alternative states of the world would be known with certainty by purchasers. As a result, the marginal cost of paying programmers the expected value of their marginal revenue product would involve no costs of deciding on the value of a contribution.[6]

It might seem that a world with zero transactions costs implies that software would have no bugs. Software bugs, however, can exist even if transactions costs are zero. Even knowing that a piece of code introduces a bug, it still can be desirable to include the code because the code is more useful for most purposes. Furthermore, it need not be the case that anyone knows how to eliminate the bug.

As a result, programmers could be paid the expected value of their marginal revenue product. Would they be? This depends on the organization of production and distribution. Coase (1937) argues that the existence of nonzero transactions costs is necessary to explain the existence of firms. Others have suggested other reasons for the existence of firms. In the end though, it seems that given the thorough definition of transactions costs above, zero transactions costs imply that market

[6] I use the term "marginal revenue product" because it includes monopoly and perfect competition in the market for the output.

transactions would be sufficient for virtually any coordination of activity.[7] If firms are defined as coordination of activities by central direction rather than market prices, zero transactions costs leave little role for any useful distinction between an employee and an independent contractor.

Furthermore, the relatively small amount of physical capital used by programmers provides little explanation for the existence of firms. In fact, the principle that people most affected by something should bear the cost suggests that programmers are the most efficient owners of their hardware.

In short, with zero transactions costs, market transactions would be sufficient to coordinate programmers' activities in producing software. Buyers would contract with the programmers directly and competition between programmers would reduce their returns to the expected value of their marginal revenue product.[8] This competition would occur before development and the marginal cost of adding another buyer would be approximately equal to the marginal benefit. This solution mimics the one proposed by Demsetz (1968) for utilities.

Some programmers would be both consumers and producers of any given program. Indeed, programmers both consuming and producing a program might not be at all unusual. To the extent that programmers who write and use a program are similar to other buyers of a program,

---

[7] The purpose of this paper is not to make a contribution to the reason for firms. If there seems a compelling reason for firms with zero transactions costs, then suppose that it is unimportant in this application. Barzel (1997, Ch X) provides a clear summary of existing theories. For example, Barzel (1997, Ch. X) suggests that guarantees backed by equity are sufficient reason for firms. Most software, however, provides no guarantees and anything more than minimal support is provided for a fee.

[8] It is not necessary that there is a natural monopoly in software even given the relatively low distribution cost compared to production cost (Alchian 19xx; Demsetz 1968).

programmers who use the program are in a better position to estimate the value of improvements to consumers than are others.[9]

Zero transactions costs also imply that the cost of enforcing any license conditions is zero. With zero cost of enforcing license conditions, there would be neither pirating of software nor costs of enforcing any other license conditions. On the other hand, there is good reason to think there would be no royalties; efficient contracts written before production would set royalties to zero.

B. *Positive Transactions Costs*

Transactions costs, of course, are positive. It is expensive to coordinate transactions among millions of buyers. It is expensive for users to assess the value of software prior to substantial investments in the software itself and setting it up. It is expensive to estimate the value of any individual programmer's marginal product. Payments to individuals equal to the value of their marginal product are important to induce the correct output because monitoring costs are positive and programmers' opportunity cost is positive.

What aspects of software production are conducive to open source development compared to development by a firm?

The discussion above suggests key elements of an answer to this question. Users of software programs can be capable of writing at least parts of the program. Many software programs can be broken into individual pieces; programmers do not have to understand all aspects of every associated program in order to work on one piece. There are substantial diseconomies of scale,

[9] For no self-evident reason, this is called "eating your own dog food." The phrase is said to have originated at Microsoft.

which creates an advantage of breaking up a program into pieces. Given these observations, it can be possible for programmers to write parts of a program, which I shall call processes.

Why would programmers write individual parts of a program? If, compared to other users, a programmer has a higher marginal benefit, a lower marginal cost or both of writing a process, then it can be worthwhile for a programmer to write a process.

Even so, why would a programmer make his work available to others? A programmer can make an improvement, keep it to himself and possibly collect monetary payments for himself. Consider a simple case — a programmer has improved a routine in a process. There must be a marginal benefit or the programmer will not make it available to others.[10] Recall the proposition that software continually is under development. The programmer has no reason to suppose that his revision of the routine is the last one that ever will be made. If the programmer does not make the improvement available to others, then later improvements will be made to the routine without his improvements. When these later improvements are made, the programmer will have to synchronize his changes with these later changes. These synchronization costs can be substantial. Hence, there is a positive return to making improvements available to others as long as others do so. In other words, it can be consistent with a Nash equilibrium to make improvements and to make those improvements available to others.

[10] An answer that they would do so out of generosity trivializes the answer for the same reason that it is a trivial answer to suppose that programmers write open source software as a hobby.

Furthermore, there are network effects associated with synchronization. The larger the number of other users who may be altering a routine, the greater the value of making improvements available to others because others are more likely to alter the routine later.[11]

There is one very important issue that has not been examined. Who will coordinate the activities of the programmers and why? It is possible for a programmer to write what he believes is an improvement when it is not. It also is possible for two programmers to write actual improvements to the same routine that are mutually exclusive. Someone must decide whether an apparent improvement is real and someone must decide what improvements to include.

Coordination has been provided by individuals who begin development of one or more processes and oversee it (Stallman 1999; Torvalds 1999; Wall 1999.) The costs are obvious. What are the pecuniary gains from coordinating others' activities? While Richard Stallman emphasizes his nonpecuniary motivation for writing free software and he almost surely has not maximized his wealth by his career choice, he has received payments related to this work; Linus Torvalds' employment opportunities were expanded by his work on Linux; and Larry Wall has benefitted by royalties from publications related to Perl.[12] In short, each of these people has achieved prominence from their work that provides monetary payments related to the success of these projects. These payments provide returns that are not imposed as royalty fees associated with the distribution of the software itself and that do not require bearing costs of enforcing licenses.[13]

---

[11] Even if writing free software were just a hobby, these synchronization costs are an important difference between writing software and other hobbies.

[12] There is nothing in economics that requires that people maximize their wealth. Nonpecuniary returns from work are important. Otherwise, few would be professors of economics.

[13] Dyson (1997) provides an excellent discussion of these and related issues.

These returns from informing others about software, which are a positive function of the number of users, at least partly offset the increasing costs of coordination with more users and programmers.

What about the activities by lieutenants, those just below the overall coordinator, or committees as apparently developed for Apache? These people's names are not commonly read in the press. These programmers affect the direction that development takes more than those who merely write processes. Programmers with a higher expected marginal gain from determining direction of development will find such positions attractive. In addition, these programmers may be able to benefit by informing others about the software, either by royalties for publications or by employment opportunities.

IV. Conclusion

Writing open source and free software can be understood as a response to a set of benefits and costs faced by agents. Writing open source software does not have to be either altruistic or a hobby.

Whether software is written by programmers working for a single firm or is distributed as open source or free software depends first on whether the software is used by people who are capable of altering the software. The similarity between the characteristics that matter to all users and to those who are capable of altering the software affects the likelihood of development as open source software compared to proprietary software. The more similar the interests of programmers and other users, the more likely is open source development compared to proprietary software. Conversely, the more divergent the differences in programmers' marginal benefits and costs of writing various parts of the software, the more likely is open source development. If there are no such differences,

then programmers will collaborate on the software until the marginal benefit equals the marginal cost but, other things the same, that point will be reached sooner than when marginal costs and benefits vary across programmers. Hence, proprietary development is more likely to be superior.

In terms of pricing to users, open source software has an advantage over proprietary software. Open source software can distributed at the marginal cost of distribution and support can be provided at that marginal cost. There are no royalty payments built into the software.

In terms of eliciting effort by programmers, open source software has a disadvantage compared to proprietary software. Programs will be written that are useful to programmers, who need not have the same interests as other users of a program. In fact, programmers' comparative advantage at writing programs almost surely means that they do not use a computer in the same way as do users who do not program. Furthermore, many programmers working on pieces of a program will not receive any payment in exchange for improvements made. The programmer will make improvements until the marginal benefit to him equals the marginal cost. While the programmer will distribute the improvement to others because synchronization costs are positive, there is no reason to think that the amount of effort will be the same as when payment is received for the effort.

That said, it is possible that open source can be more efficient than proprietary development alone. There is nothing in the open source or free software license that prohibits firms from adding capabilities that make software more useful to non-programming users.

REFERENCES

Alchian, Armen A. 19xx. "Costs and Output."

Barzel, Yoram. 1997. *Economic Analysis of Property Rights*. Second edition. Cambridge: Cambridge University Press.

Behlendorf, Brian. 1999. "Open Source as a Business Strategy." In DiBona, *et al*., pp. 149-70.

Bradner, Scott. 1999. *The Internet Engineering Task Force*. In DiBona, *et al*., pp. 47-52.

Coase, Ronald H. 1937. "The Nature of the Firm." *Economica* 4 (3), pp. 386-405.

Demsetz, Harold. 1968. "Why Regulate Utilities?" *Journal of Law and Economics* 11 (1), pp. 55-66.

DiBona, Charles, Sam Ockman and Mark Stone. 1999. *Open Sources: Voices from the Open Source Revolution*. Cambridge: O'Reilly & Associates Inc.

Dyson, Esther. 1997. *Release 2*.0. New York: Broadway Books.

Raymond, Eric S. "The Cathedral and the Bazaar." 1998. *First Monday* 3 (March). Available at http://www.firstmonday.org/issues/issue3_3/raymond/index.html on May 24, 1999.

Salus, Peter H. 1995a. *A Quarter Century of UNIX*. Reading, Massachusetts: Addison-Wesley Publishing Company.

Salus, Peter H. 1995b. *Casting the Net: From ARPANET to INTERNET and beyond...* Reading, Massachusetts: Addison-Wesley Publishing Company.

Shapiro, Carl, and Hal R. Varian. *Information Rules*. 1999. Boston: Harvard Business School Press.

Stallman, Richard. "The GNU Operating System and the Free Software Movement." In DiBona *et al*., pp. 53-70.

Torvalds, Linus. 1999. "The Linux Edge." In DiBona *et al*., pp. 101-11.

Wall, Larry. 1999. "Diligence, Patience, and Humility." In DiBona et al., pp. 127-47.

Whinston, Andrew B., Dale O. Stahl and Soon-Yong Choi. 1997. *The Economics of the Internet*. Indianapolis: Macmillan Technical Publishing.