1. A brief history of Open Source Software (OSS)

Free/open source software (F/OSS) has its roots from near the beginning of computing and is typically free while providing users with source code that is usually shared via the internet and can be adjusted for users' own needs. In the 1960s, while using computers for their work, researchers had to share software code because commercial software was not available (Moon & Sproull, 2002). The three major eras in the history of OSS are discussed here.

1.1 First era

In the 1960s, key features of operating systems and the Internet were developed by computer scientists working in Academic settings such as Berkeley and MIT. Corporate research facilities like those at Bell Labs and Xerox also played an important role in developing these essential components of modern computing. A significant milestone of this era was the development of the UNIX operating system by a group of AT&T employees at Bell Labs. The popularity and growth of UNIX was significantly aided by its adoption in academia by virtue of its source code being freely available. Several independent computer scientists and programmers contributed to building software that eventually became part of the UNIX system.

1.2 Second era

In the early 1980s, AT&T, motivated by the huge growth of UNIX, began enforcing its purported intellectual property rights on the software. In response to this threat, Richard Stallman - then a young computer scientist at MIT artificial

intelligence - founded the Free Software foundation and introduced the General Public License (GPL) for a computer operating system called GNU. GNU was envisioned as a F/OSS operating system similar in design to UNIX but governed by the terms of the GPL, which would essentially ensure that any additions and enhancements to the software were distributed under the same GPL license. These terms made the GPL viral in nature. Stallman saw this as essential for ensuring that F/OSS remains free and open source and is constantly improved through global collaboration. However the GNU operating system was still lacking an essential component – a *kernel*.

1.3 Third era

In 1991, Linux Torvalds developed the LINUX kernel. He released the kernel source code under a GPL license and Linux became a part of the GNU/Linux operating system, one of the most popular and successful open source software projects today. Another significant piece of open source software was developed in 1994 by Brian Behlendorf. This was the Apache web server. As of May 2011 Apache was estimated to serve 63% of all websites and 66% of the million busiest.[1] More recently OSS technologies such as Sendmail, PERL, PHP, Python have become indispensible components of computing and the Internet.

2. Valuation of OSS

[1] "May 2011 Web Server Survey | Netcraft", n.d., http://news.netcraft.com/archives/2011/05/02/may-2011-web-server-survey.html.

2.1 GNU/Linux

One of the earliest attempts at market valuation of a major OSS project was made by David A. Wheeler in 2001.[2] He used Red Hat Linux 7.1 as a representative GNU/Linux distribution. He based his estimate on analyzing the amount of source code in GNU/Linux - "It would cost over $1 billion ($1,000 million - a Gigabuck) to develop this GNU/Linux distribution by conventional proprietary means in the U.S. (in year 2000 U.S. dollars). Compare this to the $600 million estimate for Red Hat Linux version 6.2 (which had been released about one year earlier). Also, Red Hat Linux 7.1 includes over 30 million physical source lines of code (SLOC), compared to well over 17 million SLOC in version 6.2. Using the COCOMO cost model, this system is estimated to have required about 8,000 person-years of development time (as compared to 4,500 person-years to develop version 6.2). Thus, Red Hat Linux 7.1 represents over a 60% increase in size, effort, and traditional development costs over Red Hat Linux 6.2. This is due to an increased number of mature and maturing open source / free software programs available worldwide."[3]

The following table compares the size of Red Hat Linux 7.1 with other large software systems in terms of SLOC.

---

[2] "More than a Gigabuck: Estimating GNU/Linux's Size", n.d., http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html.
[3] Ibid.

Product	SLOC
NASA Space Shuttle flight control	420K (shuttle) + 1.4 million (ground)
Sun Solaris (1998-2000)	7-8 million
Microsoft Windows 3.1 (1992)	3 million
Microsoft Windows 95	15 million
Microsoft Windows 98	18 million
Microsoft Windows NT (1992)	4 million
Microsoft Windows NT 5.0 (as of 1998)	20 million
Red Hat Linux 6.2 (2000)	17 million

Red Hat Linux 7.1, at over 30 million physical SLOC, is much larger than these systems. Wheeler computed the estimated cost of developing Linux using the COCOMO (Boehm, 1981) as follows.[4]

Total Physical Source Lines of Code (SLOC) = 30152114

Estimated Development Effort in Person-Years (Person-Months) = 7955.75 (95469)

(Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))

Estimated Schedule in Years (Months) = 6.53 (78.31)

(Basic COCOMO model, Months = 2.5 * (person-months**0.38))

Total Estimated Cost to Develop = **$ 1074713481**#

#(average salary = $56286/year, overhead = 2.4).

[4] Ibid.

2.2 OSS

In 2008, Black Duck Software, using a similar COCOMO model, estimated the cumulative cost of all OSS software on the Internet. They estimated that it would take **$387 billion** to develop the available OSS by traditional proprietary means in year 2008 dollars. Their COCOMO estimates are as follows.

The basic COCOMO equations take the form:

E=ab(KLOC)^bb

D=cb(E)^db

P=E/D

where E is the effort applied in person-months, D is the development time in chronological months, KLOC is the estimated number of delivered lines of code for the project (expressed in thousands), and P is the number of people required. The coefficients ab, bb, cb and db are given in the following table.[5]

COCOMO coefficients for OSS valuation

[5] "Details - Estimating the Development Cost of Open Source Software | Black Duck Software", n.d., http://www.blackducksoftware.com/development-cost-of-open-source-details.

Software project	ab	bb	cb	db
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

COCOMO calculations

COCOMO Cost Estimates	
ab	2.4
bb	1.05
KLOC	4,932,000
E (P-Months)	25,579,112
E (P-Years)	2,131,593
Average salary	$ 75, 662

Overhead (wrap rate)	2.40
Total Estimated Development Cost	**$387,073,763,266**

3. Free Software and Imputation

3.1 Imputed Value

The economic theory of imputation says that the value of a good is more a matter what the buyer is willing to pay than the cost the seller incurs to create it. Carl Menger, the founder of the Austrian school of economics, first expounded the theory of imputation. This theory is contrary to the labor theory of value maintained by classical economists such as Adam Smith. The value of OSS calculated using COCOMO confirms with the classical approach of determining factor prices using the cost of labor and other resources. Imputation suggests that the value was not made up of the factors that made up a good; instead, it was made up of the most valuable use that the last good could be put to—the marginal utility of the finished good. In the case of F/OSS it is not easy to determine this imputed value since F/OSS is essentially a higher order good, unlike a consumer good whose value can be determined by its optimum sales price in the market. F/OSS itself serves as a factor of production in other large and complex software projects, both open source and corporate and powers nearly every large website or software system in the world. It

is the cumulative marginal utility of all these secondary consumers of F/OSS that would determine its value. This implies that F/OSS is highly valuable despite being free.

3.2 Motivation and utility

While it is clear that consumers of F/OSS derive considerable utility through using free software to run their own software systems, the question still remains as to what is the value of F/OSS to programmers and hackers who actually build free software without expecting any financial benefits or intellectual property rights. One possible explanation is given by Eric. S. Raymond, a popular hacker in the OSS community - "The 'utility function' Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers. ... Voluntary cultures that work this way are not actually uncommon; one other in which I have long participated is science fiction fandom, which unlike hackerdom has long explicitly recognized 'egoboo' (ego-boosting, or the enhancement of one's reputation among other fans) as the basic drive behind volunteer activity." This 'egoboo' phenomenon may be one of the motivations for tenured academics to publish papers.[6]

[6] "The Cathedral and the Bazaar", n.d., http://www.catb.org/~esr/writings/homesteading/cathedral-bazaar/.

Additionally, individuals participating in OSS development projects can increase their potential wages and income in the future due to development of market signals (Spence, 1974) and individual reputation.[7]

4. Why use F/OSS?

Richard E. Hawkins examines why computing projects use F/OSS in his paper titled "Netnomics" (2004), and the answer may simply be "Well, why not?" For a computing project, the costs can be divided into hardware costs for the purchase of equipment E, the purchase of the software at a price B, internal administrative costs A, external support costs S, and down time costs D to the firm from times when the system is unavailable due to failure or while waiting for repair of hardware or software, for a total cost C.[8]

This can be expressed as follows.

Cost to consumer, $C = E + B + A + S + D$

When deciding between alternative software systems, the consumer's problem is to minimize the following.

$\min C = A + S + D$

[7] Chong Ju Choi, Sae Won Kim, and Shui Yu, "Global Ethics of Collective Internet Governance: Intrinsic Motivation and Open Source Software," *Journal of Business Ethics* 90, no. 4 (December 1, 2009): 523-531.
[8] Richard E. Hawkins, "The economics of open source software for a competitive firm," *NETNOMICS: Economic Research and Electronic Networking* 6, no. 2 (August 2004): 103-117.

With A << C (administrative costs become insignificant as project members become familiar with a software system or already know how to manage it), there is no compelling reason to use or not use open source software.

5. Why give software away for free?

R. E. Hawkins, in the same paper, examines why companies may choose to give software away as F/OSS. An important example in analyzed in the paper is Apple's Darwin, the operating system that runs underneath its OS X.

5.1 Costs and strategic concerns for the firm

Hawkins models an economics strategy for F/OSS firms - Open source is not an exception to the laws of economics, recent dotcom hysteria not withstanding. To continue its existence, a firm must incur expenses, which in the long run must be less than its revenues. Relying upon open source software will incur expenses, possibly even greater than those incurred in a proprietary solution. The goal of the firm is not to maximize revenue, nor to minimize expense, but to maximize the amount by which revenues exceed expenses.

Of particular concern to the firm is the possibility of decreased development costs with open source software, both in development and maintenance: if the source is open and used by others, the firm can take advantage of the development work of others. Weighing against this benefit is the ability of other firms to benefit as well. This introduces strategic considerations from the firm: how will other firms react to the firm's release of the source, and whether the losses L resulting from sales lost to

other firms will be greater than the savings in development costs D. The costs to the firm can then be represented as follows.

$C = D + L$[9]

5.1 Strategic games

Regarding Darwin, Hawkins says, "Darwin is derived from FreeBSD, NetBSD, and the Mach microkernel from Carnegie Mellon University, all of which are under public licenses. It also has technology from NeXT, which was purchased by Apple. It was entirely within Apple's ability to retain the changes made. Nonetheless, Apple released Darwin under a public license. Furthermore, Apple fed large numbers of bug fixes back to NetBSD, in spite of the absence of any obligation to do so. Apple's motivation may be seen in its announcement that it will be "synchronizing" Darwin to FreeBSD: by turning over its own changes, even those that would give it some competitive advantage, Apple's product is "automatically" maintained. If Apple were to make a private change in the code, it would benefit in the short run. In the long run, another change would be made in the same sections of the "other" source base. As Apple's code would now be different, the change could not be directly made; that is, the opportunity cost of keeping the changes is forgoing the external maintenance. Apple has no interest in Darwin in and of itself; it is a necessary component for its OS X product – and a component that Apple tried and failed to develop at least twice."

[9] Ibid.

The following game demonstrates why open source was the optimum strategy for Apple in the case of Darwin.

Simple game for firm and public.

Firm		Public	
		Ignore	Contribute
	Closed	(0, 0)	(0, 0)
	Open	(5, 0)	(7, 5)

This table shows the game for a hardware firm, which needs a program as part of its product and faces no direct competitor able to benefit from the code. If the firm chooses a proprietary design, the cost is 10 (payoffs are reported relative to this amount). However, there is a program already available, as in the case of Apple and Darwin, which can be used by the firm, reducing the cost by 5. With no competitor to take advantage of the code, there is no incentive for the firm to not release the changes (as is the case for a viral license). The only other player is the public, which can either ignore the code, or can contribute and release changes. Between these changes and the fact that the public will provide a portion of the regular maintenance of the software, the firm obtains an additional payoff of 2 when the public chooses to contribute. In this case, the open source option is a dominant strategy – D is lower for the firm regardless of the choice made by the public.[10]

A similar game can be used to analyze the optimum licensing strategy for the firm. This is demonstrated in the following table.

[10] Ibid.

License choice for the firm.

Firm	Public	
	Ignore	Contribute
Closed	(0, 0)	(0, 0)
Public	(5, 0)	(7, 5)
Viral	(4, 0)	(5, 4)

In this game the public license is the dominant strategy. BSD and MIT are examples of public licenses. These licenses are less restrictive than public licenses such as the GPL.

By using a public license, Apple was entitled to keep the code changes, but chose to forgo doing so for strategic reasons. If Apple had used a viral code base, however, not only would it have been forced to release the Darwin code, but would also have had to either take great care to insure that no portion of OS X was a "derived work" of Darwin within the meaning of copyright law (which might or might not have been possible), or release the source to OS X.[11] This explains why the utility for the firm is lower (5) in the case of a viral license as compared to a public license (7).

The viral license can only be used by a subset of the public that could use the public license. This is necessarily true, as the use of public code in a viral project is possible with all major licenses in use today, while the converse is not true. These

[11] Ibid.

constraints lower the utility of the code for both the firm and the public in the case of viral licenses as compared to public licenses.

However, public licenses such as GPL and LGPL are far more popular in the industry than public licenses. This is a result of the fact that in real world game situations very few firms use software systems that have no competitor in the market. The following game demonstrates how viral licenses are the optimum strategy when a firm has a competitor for its software in the market.

Firm with license choice and potential competitor (firm, public, competitor).

Firm		Public/market competitor					
		Ignore/ ignore	Contribute/ ignore	Ignore/ hoard	Contribute/ hoard	Ignore/ contribute	Contribute/ contribute
	Closed	(0, 0, 0)	* * *	* * *	* * *	* * *	* * *
	Public	(5, 0, 0)	(7, 5, 0)	(2, 0, 3)	(4, 5, 4)	(3, 0, 1)	(4, 7, 1)
	Viral	(4, 0, 0)	(5, 4, 0)	* * *	* * *	(4, 0, 1)	(6, 6, 1)

The viral license is the dominant strategy for the firm in this game since it essentially eliminates the possibility of a competitor using the firms code and hoarding its enhancements and modifications, which could result in larger market share for the competitor's product.

In this game, the cost to the firm of 1 for monitoring a viral license is kept. The firm is assumed to benefit by 2 for the contributions, if any, from the public and by 1 from contributions by the competitor. However, if the firm must face the competitor in the market places, it suffers losses L of 3 due to lost sales, while it faces a loss of only 1 if it can incorporate the competitor's changes into its own product. The public receives a gain of 1 if contributions are made by the competitor.

The ability to acquire the software is valued by the competitor as 3 if it keeps its changes private, with an additonal 1 for contributions by the public, but at only 1 if it releases them, as its product could be obtained at no cost. The public values the contributions by the competitor at 2. With a public license, the competitor may choose to either "hoard" any changes it makes for its own use, or contribute them back to the general project.

In this game, the firm's best choice is a viral license. If it chooses a public license, the market competitor can take the software, inflicting losses L, with no offset to the firm. The viral license forces the competitor to release changes, which (in this case) yields the same payoff for the firm as if the competitor had ignored the software.[12]

6. Business Models

Now that we understand the economics of why firms use and develop F/OSS and which OSS licenses are optimum in the market, we finally looks at some of the popular business models for F/OSS.[13]

The following table is adapted from Raymond (2000b).

[12] Ibid.
[13] "The Cathedral and the Bazaar."

Name	Business model	Example
Loss-leader / Market positioner.	Use open source software to maintain a market position for a related proprietary software product.	Netscape's open source Mozilla web browser and proprietary server software.
Widget frosting.	Sell hardware with open source driver software.	Apple's MacOS X.
Give away the recipe, open a restaurant.	Distribute open source software and sell service and support contracts.	Red Hat.
Accessorising.	Sell accessories for open source software such as documentation.	O'Reilly and Associates.
Free the future, sell the present.	Sell closed source software with a license that makes it open source after a specified time period.	Aladdin's Ghostscript.
Free the software, sell the brand.	Sell other developers a brand that certifies their implementation of your open source technologies is compatible with all others who use the brand.	Sun's StarOffice.
Free the software, sell the content.	Develop an open source product that receives proprietary content that the firm sells.	N/A.

7. Conclusion

F/OSS is a valuable commodity in the market, whether one considers its value as imputed or not. Despite the common assumption that F/OSS developers are altruists giving away valuable software for free, there are established economics theories that can explain that individuals and firms are maximizing some utility function while using and creating OSS. Strategic games can be used to analyze and explain why firms give away software as open source and for free. Economics can also explain why one form of OSS licenses may be better than others in a given market situation. Public licenses such as BSD are a good strategy in the absence of a direct competitor, but viral licenses such as GPL prevent competitors from hoarding software enhancements and are a good strategy in competitive markets. F/OSS is not only about releasing software for free along with its source code but there are several business models that can be built around a F/OSS system.

References

Choi, Chong Ju, Sae Won Kim, and Shui Yu. "Global Ethics of Collective Internet Governance: Intrinsic Motivation and Open Source Software." *Journal of Business Ethics* 90, no. 4 (December 1, 2009): 523-531.

"Details - Estimating the Development Cost of Open Source Software | Black Duck Software", n.d. http://www.blackducksoftware.com/development-cost-of-open-source-details.

Hawkins, Richard E. "The economics of open source software for a competitive firm." *NETNOMICS: Economic Research and Electronic Networking* 6, no. 2 (August 2004): 103-117.

"May 2011 Web Server Survey | Netcraft", n.d. http://news.netcraft.com/archives/2011/05/02/may-2011-web-server-survey.html.

"More than a Gigabuck: Estimating GNU/Linux's Size", n.d. http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html.

"The Cathedral and the Bazaar", n.d. http://www.catb.org/~esr/writings/homesteading/cathedral-bazaar/.