About the Authors

Samir Chopra
Assistant Professor
Department of Computer and Information Science
Brooklyn College

Samir Chopra is an Assistant Professor in the Department of Computer Science at Brooklyn College of the City University of New York. Samir earned a BA in Mathematical Statistics from Delhi University (1984), an MS in Computer Science from the New Jersey Institute of Technology (1990) and a PhD in Philosophy from the City University of New York (2000). His current research concentrates on the philosophical foundations of artificial intelligence and the politics of technology. With Scott Dexter, he is working on a book on the philosophy of free software, forthcoming from Routledge.

Scott Dexter
Associate Professor
Department of Computer and Information Science
Brooklyn College

Scott Dexter is an associate professor of Computer and Information Science at Brooklyn College of the City University of New York. He received his B.S. in Mathematics and Computer Science from Denison University and his M.S. and Ph.D. in Computer Science and Engineering from The University of Michigan. His research interests include network and multimedia security, distributed computing, computer science pedagogy, and the history, politics, and economics of technology. He has been an invited speaker at meetings in Philadelphia, Toronto, Prague, and the People's Republic of China, and is co-author of an introductory programming textbook.

Contact Details

Samir Chopra
Assistant Professor
Department of Computer and Information Science
Brooklyn College
2900 Bedford Avenue
Brooklyn, NY 11210

schopra@sci.brooklyn.cuny.edu

Phone: +1 718 951 4139 Fax: +1 718 951 4842

Scott Dexter
Associate Professor
Department of Computer and Information Science
Brooklyn College
2900 Bedford Avenue
Brooklyn, NY 11210

Phone: +1 718 951 3125 Fax: +1 718 951 4842

The Political Economy of Open Source Software

"For us, open source is capitalism and a business opportunity at its very best."

— Jonathan Schwartz, President and Chief Operating Officer, Sun Microsystems

Main Description

We investigate the political economy of free and open source software and its location vis-à-vis Marxist critiques of capitalist modes of production. In particular, we examine the extent to which open source software invokes or revises traditional notions of property and production. We hypothesize that open source software is not anti-capitalist but instead is an evolutionary step towards what has been termed 'late capitalism'. We produce a critique of open-source development, arguing that while it manifests a number of anti-capitalist tendencies, it is fundamentally aligned with post-modern capitalist development models.

Short Description

A critique of open-source software development, arguing that while it displays anti-capitalist tendencies, it is aligned with post-modern capitalism.

Keywords

Open source software; free software; political economy.

What is Open Source Software?

The term *open source software*, to which *free software* is closely related,[1] refers at once to a particular approach to distributing software and to the dramatic implications this has for the process of developing software.

The software that we run on our computers is represented as a sequence of instructions for the computer to execute; these instructions are represented, in a fashion directly 'understood' by the computer's hardware, as 0s and 1s. These so-called *binaries* are extraordinarily difficult for humans to understand; while it is theoretically possible to determine the purpose and function of a program in binary form, it is exceedingly time-consuming and only rarely attempted. Similarly, it might be possible to modify the function of a program by modifying its binary representation, but this is again expensive and unsustainable. Instead, the vast majority of modern software is written using any of a variety of *high-*

[1] We use Free/Open Source Software, or F/OSS, to include both notions explicitly—there are important distinctions between open source and free software, hence our explicit usage of both terms when the difference is crucial

level languages, which, while difficult to interpret without a little training, nonetheless enable programmers to communicate the logic of their programs to other programmers using language and symbols that are intentionally based on natural language (usually English) and mathematics. Thus, it is reasonably straightforward for one programmer to read another's work and understand not only the function of the program but the manner in which that functionality is achieved. Automated translation programs (*compilers*) then convert this high-level representation (*source code*) into computer-executable binary code.

In the past few decades, most commercial software has been distributed in binary form only, thereby providing users with usable programs, but concealing the techniques by which these programs achieve their purposes. Source code for such programs is regarded as a trade secret, the revelation of which would have potentially disastrous effects for its corporate creator.

But there is an alternative: to distribute software *with* its source code. This is the guiding principle of F/OSS. At various times in the history of software development, in particular communities of programmers and enthusiasts, and increasingly among modern software corporations, distribution of source code has been and continues to be a fundamental practice. This distribution creates several potentials: for users to *inspect* the code of the software they use, *modify* it if they are so inclined, and—most important—send the modifications back to the originator for incorporation in future versions of the software. Allowing this form of user participation in the evolution of software has created vast and sophisticated networks of programmers, software of amazingly high quality, and new business practices.

As we discuss below, F/OSS in its modern incarnation was founded largely on an ideology of "freedom, community, and principle," with little regard for the profit motive.[i] Yet today F/OSS is making headlines daily as corporations relying on open source demonstrate remarkable success, and corporations that before hewed to closed-source distribution now open significant parts of their products. How have the political economy and cultural logic of F/OSS played into this tale of the underdog?

F/OSS and Political Economy

F/OSS is enough of a phenomenon to become the subject of numerous political, economic, and sociological studies. These studies fall mainly into three categories:

- F/OSS is a novel technology for producing software; what are the micro-economic, political, and personal dynamics that support it? (This is the focus, for example, of Eric Raymond's essays "The Cathedral and the Bazaar"[ii] and "Homesteading the Noosphere;"[iii] and Steven Weber's *The Success of Open Source*[iv]);

- F/OSS provides a social good that proprietary software can't; for example, F/OSS may be the only viable source of software in developing nations, where programming talent is available but prices for proprietary software licenses are prohibitive.[v]
- F/OSS is a threat to the industry status quo. This position has been promoted vigorously by open source advocates—most notably by Eric Raymond in his essay "The Magic Cauldron"[vi]—who argue that open source software is a new and better way of doing business: one that should, as a result of free market competition, supplant much (though not all) of the proprietary source code produced and sold today. Stakeholders in the status quo are demonstrably aware of this threat, as the so-called "Halloween Papers" leaked from Microsoft dramatically show.[vii]

In this paper we are most interested in the third aspect of F/OSS as an economic phenomenon; in particular, we try to discern how it is that a suite of production technologies can simultaneously embody radical ideals of cooperation, freedom, and social change and be an ever more widely embraced model of capitalist software production.

We first sketch a brief history of the F/OSS movement and then proceed by placing the salient characteristics and ideologies of F/OSS in dialog with theorizers of capitalism. We show that F/OSS, even its original conception, and certainly in the present moment, is a nearly archetypal post-modern phenomenon, in which inhere many of the qualities necessary to support late capitalism.

A Brief History of Software Development Technologies

The contemporary relationship between F/OSS and capital is evolving rapidly and difficult to assess. We offer a periodization of the evolution of software development technologies strongly analogous to Fordism-centered analyses of developments in the manufacturing sector. This provides a framework through which to view the economic and cultural significance of F/OSS and begins to shed light on the import of the distinction between free and open-source software. While originally orthogonal to proprietary software and its attendant capitalist culture, there is much resonance between open-source and those older models.

Stripped of nuance, the history of production technology in the manufacturing sector describes an arc from artisan craftsmanship to Fordism's division of labour and assembly lines to post-Fordist flexible production supported by modern information and communication technologies. Software development techniques have followed a similar evolution. As Weber notes, the first programmers "perceived themselves as craftspeople [and] their culture as one of artisanship as much as engineering…. [T]hey wanted to be responsible for a project from start to finish."[viii] Fordist organizational schemes were first applied to programming by the aerospace industry in the late 1950's and were subsequently taken up across the software industry. At first, this simply meant machine operators were separated both physically and organizationally from programmers—a change experienced by programmers as a significant loss of autonomy.

Subsequently, theories of the division of labour were introduced into the programming process itself, championed first by Harlan Mills[ix] and, most famously, by Frederick Brooks, in his 1975 text on software engineering, *The Mythical Man-Month*.[x] Mills and Brooks lay out principles by which the labour of creating source code itself may be divided among groups of programmers to facilitate both efficient development and high-quality code.

Parallel to these developments in the corporate programming sector, a thriving community of hobbyists, enthusiasts, and entrepreneurs was experimenting with what would come to be called personal computing. As they explored what could be accomplished with very limited hardware, their attitude towards software was that it should be shared freely to further innovation and spread the word about the growing power of personal computing.

The notion of software as a good that could be sold, as property that could be stolen, was an alien one, until in 1975 Bill Gates wrote an open letter to the Homebrew Computer Club, accusing its members of stealing his software. With Microsoft still in its infancy, Gates was only the co-author of a small (if important) program: a translator for the BASIC programming language; this translator is what allowed most hobbyists to experiment effectively with their Altair computers. In his letter, he notes that hobbyists at the time believed that, Hardware must be paid for, but software is something to share." He claimed that less than 10% of Altair owners had actually paid for the BASIC software he had written, and summarily stated, "Most directly, the thing you do is theft."[xi]

With this intervention by Microsoft, positioning software as a commodity rather than as a common good, the embryonic software industry was set on a course of Fordist production of proprietary software-for-sale, which maintained hegemony from the late 70s to the late 90s—though supplemented by a still-thriving hobbyist community that exchanged free- and shareware. It also created the possibility for programmers to be viewed as highly paid technocrats (which took on extreme proportions after the tremendous commercial potential of the Internet became apparent).

In 1985, Richard Stallman, a long-time programmer at MIT's Artificial Intelligence Lab, founded the Free Software Foundation, directly in response to the restrictions being placed on software sharing at MIT. His tangible goal was to develop a free operating system dubbed GNU, though this goal emerged from a broader philosophical and social goal to replicate and disseminate the ideals of freedom and cooperation that characterized much of 'hacker culture.' (As Stallman has often said, "Think 'free speech,' not 'free beer.'") These values, he perceived, were being steadily eroded by the increasingly proprietary nature of commercial software. Central to his objective was the practice of providing the source code of all software so that users could modify, enhance, and customize their software without restriction—as long as any distribution of a modified version also included the source code.[xii]

The Linux project (begun independently, though ultimately dovetailing with much of the Free Software Foundation's work), shared the practice of distributing source code, though in this case it was largely for the pragmatic value of having as wide as possible a range of talented programmers performing corrections and improvements which were steadily incorporated into the 'central' version.

As GNU, Linux, and a growing collection of powerful free software consistently demonstrated their superiority to proprietary commercial software—thus representing a credible threat to Microsoft's hegemony—a community of free software developers declared an "open source movement". This philosophical and tactical schism, spearheaded by Eric Raymond, had the explicit purpose of increasing the role of open source software in the business world. Claiming that the term "free software" is too confusing for business leaders (who apparently don't understand the difference between free speech and free beer), they chose the term "open source" as a putatively clearer designation.[xiii]

While the open source movement shares many philosophical and pragmatic positions with the free software movement, their views on the rights and responsibilities of users are subtly but significantly different. Most importantly, the open source movement incorporated software distribution licenses allowing for modified versions of software that was originally open to be released as closed or proprietary (and for open source software to be used in products that included proprietary components).

The F/OSS practice of distributing source code and accreting changes submitted by user-programmers marks the beginning of post-Fordist production in the software industry. The essentially Fordist practice of dividing labour among a pool of programmers is enhanced, expanded, and rendered radically flexible. The labour pool for an open source project is not limited to a group of engineers inside a company but is expanded to include literally anyone. Exploiting the Internet, the cycle of distribution and accumulation of modifications is orders of magnitude more efficient and effective than the code-sharing of the 1970s.

F/OSS and Late Capitalism

Late capitalism relies on mobilizing the powers of intellectual labour, a claim manifest in Harvey's assertion that the "flexible accumulation regime" solved a crisis for capital by moving some "absolute" surplus value, which old capitalism derived via the longer workday and lower standard of living, to "relative" surplus value by reducing costs of goods through investment (for example, in technological innovation)[xiv]. For our purposes, late capitalism refers to a cluster of related notions related to a global movement beyond Fordism. As Jameson says:

> besides new forms of transnational business… [late capitalism's] features include the new international division of labor, a vertiginous new dynamic in international banking and the stock exchanges…, new forms of media interrelationship (very much including transportation systems such as containerization), computers and automation, the flight of production to advanced Third World areas, along with… the crisis of traditional labor[xv].

This entails a necessity for information technology and a skilled work force. F/OSS, with its flexible labour force, rapid technical innovations and its reliance on technological advances, not only embodies post-Fordist ideas about production but also is embedded in a world economy in which technology plays an increasingly critical role. Indeed, developments in the F/OSS movement closely mirror many of the phenomena associated with the emergence of late capitalism.

A quick survey of other standard treatments shows that the term is also used to describe the globalization of capital, the movement from manufacturing to service based economies in the First World, the dispersion of labour due to porous borders and strengthened electronic command and control, the imposed flexibility of labour (long working days, 24-hour factories), the increasing interconnectedness of world economies, the growth of knowledge as a commodity in itself and the displacement of the marginal cost model of production. In this view, flexibility and geographic dispersal are tools through which capitalism has become more organized.

Standard analyses of late capitalism claim that knowledge is the key commodity in its markets. Harvey for instance, speaks of the limits to the accumulation or turnover of physical goods and suggests that capitalists increasingly are driven to provide ephemeral services instead[xvi]. Concomitantly, commentators on open source have noted that its programmers participate in order to trade—explicitly—on their knowledge and skills. The willingness of programmers to share their code demonstrates an understanding that their knowledge is the truly valuable commodity, not the products—understood as ephemeral—made by them. (Or, as Raymond exhorts, "give away the recipe, open a restaurant"). F/OSS programmers have turned to this model as a way of supporting themselves by selling their expertise in programming rather than relying, however indirectly, on the sales of particular products they developed.

F/OSS and Labour

Standard Marxist critiques claim workers are deprived of the surplus value associated with a product. The capitalist owns the means of production, makes the worker produce for a pittance, and sells the goods at a profitable margin. The labour value provided by the worker is denied to her: Because the worker sells her labour power to earn a living, and the capitalist owns the labour process, the product of her labour is alien to the worker.

In the nascent computer industry, when hardware vendors provided software, control of the means of production stayed with the corporate owner and hence source code was freely available. Once the two were divorced, there were new players: the software capitalists, who needed a way of retaining similar control of the software. Closing the source code was the most straightforward way of doing so. In this model then, the 'means of production' remain with the corporate owner of the software, because the worker is unable to modify the code.

We emphasize the inability of the worker to modify the software as a source of his putative alienation because software is a good that sits uncomfortably in any taxonomy of goods and products. It has been described variously as intellectual property, as art, as a manufactured good subject to engineering analysis and as an intangible. Because of its inherent modifiability—any piece of code can theoretically be extended to increase its functionality—a blockage on the means of modification is a fundamental restriction on access.

There is another entity in this picture—the *user*, who is alienated from the software as well since it must necessarily appear unknown to him. He is unable to perceive the product's infrastructure or change the product to meet his specific needs. The F/OSS model, which modulates this alienation by casting users as workers who might modify the product, is a crucial addition to the classical view of the manufacturing world.

The manufacturing model for F/OSS is a late capitalist model employing immaterial labour. The transparency of the code on a communication network such as the Internet means that the code resides everywhere, with multiple copies extant and the labour force—consisting of a vast pool of highly contingent labour—dispersed across all time zones. Workers work on separate copies of the code, writing new code or looking for flaws in newly released code, before sending suggested modifications to a central assembly point. The controls present in modern systems of quality control of code are powerful; while there is a flurry of fixes, there is strict control over what gets admitted. (As we see below, each node in this network retains the freedom to designate itself as either a replacement or an additional central assembly point).

While F/OSS relies on the "distribution of labour"—an enhanced form of the division of labour—as it throws open the gates of the virtual factory, the discipline within this factory can be as hierarchical as anything imagined by the automobile moguls of the 50's. The development of Linux, for example, is controlled by one man (Linus Torvalds) sitting astride its mountain of code. His lieutenants maintain control over the product through a rigorous system of quality control.[2] What is different about this model is that the contributions come from everywhere, from all time zones. In contrast, proprietary software blocks off participation in the production of the software. Ironically, it refuses help in its enterprise, fearing co-optation. F/OSS wants lots of users; closed-source wants lots of buyers but few users. But here users are also workers—they are also the producers of the future versions of the code. As Eric Raymond claims, open source *peer review* is the only *scalable* method for achieving high reliability and quality: this technique is the sole manner in which the energies of an army of programmers can effectively be focussed on solving the problems of creating excellent code.

The availability of technological advances to the workers (and the empowerment they bring in their wake) exerts stronger pressures of labour control on the capitalist/owner. In this mode of production, struggling against exploitation

[2] Though the possibility of rebellion against this control, through forking, is an integral part of open source development, as we see below.

is very different than struggling against exploitation in a factory. For Harvey, the disciplining of labour power is an intricate affair— a delicate mix of cooperation and cooptation. Open source shows such a mixture in its co-optation of the utopian spirit of free software model, as workers have already bought into the ideology of open source or free software production. The nature of the exploitation is subtle. While the education and flexibility of open source programmers make it harder for capitalists to control the labour force, control does exist.

F/OSS, then, solves capital's problem of exerting sufficient control over labour power to guarantee addition of value. Traditional commodity production locates the knowledge and decisions of technique outside of the worker. This is not the case in F/OSS, which brings with it a different relation of labour to the product and partially redresses the inability of capitalism to satisfactorily regulate labour to support its own propagation.

Does F/OSS represent true progress in labour relations? Is it correct to view the political economy of open source software as a substantial response to Marxian notions of alienation in its radical configuring of the relationship of worker to product? The empowered conception of the open source user as programmer and co-worker certainly changes the relationship of the worker to the product. The relationship is also different because of open-source's unique design aesthetic—a mutually modifying one, a mode of production that transforms the producer and the produced, since the workers do not work just to produce the good, but also to improve their programming skills and learn new technologies (We frequently advise students keen to improve their programming that they work on open problems in Linux).

Alienation from the end product is mitigated, in this view, because the worker can take the good with him to work on, and derive independent profit and surplus value from it. Such a freedom is embodied in programmers' right to *fork*: to take a copy of the code and to work on a separate version of the software by themselves. The production tree for the software versions splits at this point; the original product's development proceeds along one branch, the breakaway programmer's version develops along another. This is one area in which the crucial differences between free and open source software come to light: Open source licensing schemes permit the fork to be a closed one: the resulting product need not be open source.

But what does the freedom/right to fork actually entail? While the technical rationale for the right to fork is protection against the incompetence of one set of code maintainers, it also preserves the spirit of entrepreneurship: an open source worker could make something of the code by herself. In F/OSS, the source code is freely available (though still under license and hence not in the public domain), and the means of producing copies of the software are simple data-recording devices. A worker could leave the 'factory' one day with a copy of the software—

with no legal approbation attached to this act—and commence sales of the software at any price he (and the market) sees fit.

Typical vendors for free and open source software, such as Red Hat and SuSE, are not selling software, however. They sell services, which the worker cannot expropriate. Depending on the nature of the particular software licensing scheme in play, the worker might not be able to leverage this freedom in an economically advantageous fashion.

Licensing, Labour, and Capital

Software licensed under the GPL (the GNU Public License—the paradigmatic free software license) with its so-called 'viral clause' must remain under this license in perpetuity: if a worker were to make changes to a copy of the software, then decide to distribute the new code, he must do so under the terms of the GPL. But the original 'corporate' owner could then integrate the new code into the old code base and continue selling the product as before. Because the owner in such a scenario commands market share by virtue of so-called value added services (such as support, documentation and antecedent user community), the owner is in a position to elbow out the new software product developed by the 'renegade' developer.

Alternatively, under the terms of licenses such as the Berkeley Standard Distribution (BSD) license, a programmer is allowed to modify open source code, then market the resulting product as closed source. The original 'owner' would not have access to the proprietary code and would not be able to make changes to it (or even integrate it into old copies of the software) without stringent licensing compensation to the developer). Inverting the corporate scenario, this means that an independent programmer's code could be used by any other entity, who could make modifications and then release the code as proprietary. The original unmodified code would still be open – the corporate modifications would be the only part kept secret, but such secrets are not trivial.

Both these scenarios are plausible. The ways in which these would/could play out are dependent on the sociological factors at play in the open source community. While the freedom to fork exists, there exists too, a tremendous cost in trying to actually break free of the original project, for who would join the breakaway programmer (whether corporate or individual)? The reputation of the breakaway and the judgment of him by his peers would be crucial factors in this decision.

Still, what makes open source attractive to capital is the potential to convert to closed source later (and the possibility of drawing upon the technical skills of a motivated worker force). Free software remains anti-capitalist in a crucial sense, since it is willing to sacrifice some technical/economic good in order to preserve an intangible value. The restrictions that the GPL forces upon its licensees sometimes make it harder to reach a technical solution—especially if that solution involves combining proprietary and free software in the same product. Raymond has tried to make an argument for open source based *only* on engineering and

economic factors: quality, reliability, cost and choice. This argument is a considerable distance from Stallman's altruistic notion of sharing, a determinedly anti-capitalist notion.

Conclusion

The original motivation for the open source definition (and its attendant movement away from the Free Software licensing scheme) was the concern that the word 'free' was misleading and unattractive to potential corporate supporters. But as Stallman's speech/beer refrain shows, there simply is no confusion when we say 'free speech'—no one imagines we are giving away speech for free. Stallman's choice of terms reflects the movement to (re)claim software as a general public good rather than a commercial good. That is, the "confusion" that the open source movement invokes as it seeks to create distance from the free software movement and seduce corporate interests was in fact sowed by a long campaign on the part of those very corporate interests to privatize a historically public resource.

Bibliography

Brooks, Frederick Brooks. *The Mythical Man-Month: Essays in Software Engineering*. Boston: Addison-Wesley 1995.

Dibona, Chris. Ed. *Open Sources: Voices from the Open Source Revolution*. Cambridge: O'Reilly Open Source, 1999.

Harvey, David. *The Condition of Postmodernity*. Cambridge: Cambridge University Press, 1989.

Jameson, Fredric. *Postmodernism, or, The Cultural Logic of Late Capitalism*. Durham: Duke University Press, 1991.

Lessig, Lawrence. *Free Culture: How Big Media Uses Technology and the Law to Lock Down Culture and Control Creativity*. New York: Penguin, 2004.

Levy, Stephen. *Hackers: Heroes of the Computer Revolution*. New York: Penguin, 1994.

Mills, Harlan D. *Software Productivity*. Boston: Little, Brown and Co., 1983.

Moody, Glyn. *Rebel Code: Linux and the Open Source Revolution*. New York: Basic Books, 2002.

Raymond, Eric. *The Cathedral and the Bazaar*. Cambridge: O'Reilly Books, 2001.

Stallmann, Richard. *Free Software, Free Society: Selected Essays of Richard Stallman*. Cambridge: Free Software Foundation, 2002.

Vaidyanathan, Siva. *Copyrights and Copywrongs: The Rise of Copyright and How it Threatens Creativity*. New York: New York University Press, 2003.

Weber, Steve. *The Success of Open Source*. Cambridge: Harvard University Press, 2004.

[i] Stallman, Richard. "The GNU operating system and the free software movement." In *Open Sources: Voices from the Open Source Revolution*. Sebastopol: O'Reilly & Associates, 1999, page 70.
[ii] Raymond, Eric S. "The cathedral and the bazaar." http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/
[iii] Raymond, Eric S. “Homesteading the noosphere." http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading/
[iv] Weber, Steve. *The Success of Open Source.* Cambridge: Harvard University Press, 2004.
[v] For online discussions, see: http://www.linuxjournal.com/article/6049 or http://www.bytesforall.org
[vi] Raymond, Eric S. "The magic cauldron." http://www.catb.org/~esr/writings/cathedral-bazaar/magic-cauldron/
[vii] Open Source Initiative, "The Halloween documents." http://www.opensource.org/halloween/
[viii] Weber, page 25.
[ix] Harlan Mills, "Chief programmer teams: techniques and procedures (1970)," Reprinted in *Software Productivity*, Harlan Mills. New York: Dorset House Publishing, 1988.
[x] Brooks, Frederick. *The Mythical Man-Month.* Reading: Addison-Wesley, 1975.
[xi] Levy, Stephen. *Hackers*. New York: Penguin, 1994.
[xii] Moody, Glyn. *Rebel Code: Inside Linux and the Open Source Revolution*. New York: Basic Books, 2002.
[xiii] http://www.opensource.org
[xiv] Harvey, page 120.
[xv] Jameson, Fredric. *Postmodernism, or, The Cultural Logic of Late Capitalism.* Durham: Duke University Press, 1995, page XIX.
[xvi] Harvey, David. *The Condition of Postmodernity*. Cambridge: Cambridge University Press, 1989, page 159.