Introduction

(Introduction, *Decoding Liberation: The Promises of Free and Open Source Software*, Routledge, July 2007)

By:
Samir Chopra and Scott Dexter {schopra, sdexter}@sci.brooklyn.cuny.edu
Department of Computer and Information Science
Brooklyn College of The City University of New York
2900 Bedford Avenue
Brooklyn, NY 11210

Technological artifacts of the past consisted only of hardware: engines, motors, pumps, levers, switches, gears. To control the hardware was to control the technology. Hardware is expensive to acquire and maintain, so technology was invariably controlled by large economic entities--states, then corporations. Concerns about social control invariably addressed control of technology; Marx's concerns about the control of the means of production were focused on the hardware that both crystallized and generated capitalist power.

The 20th century brought a new form of technology, one in which hardware and control are explicitly separated. The means of production no longer inhere solely in hardware; control is transferable, distributable, plastic, and reproducible, all with minimal cost. Control of technology may be democratized, its advantages spread more broadly than ever before. The reactionary response to this promise is an attempt to embrace and coopt this control to advance entrenched social, economic, and political power. It is this reaction that free software resists.

* * *

The software that runs on our computers is 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. Programs in this form, called *binaries* or *executables*, are extraordinarily difficult for humans to understand. While it is theoretically possible to determine the purpose and function of a program in binary form through *reverse engineering*, 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 too is unsustainably expensive. Instead, the vast majority of modern software is written using a variety of *high-level languages*.

Automated translation programs (*compilers*) then convert these high-level programs (*source code*) into executable binary code. Programs in high-level languages, while difficult to interpret without training, enable programmers to communicate their design logic to other programmers using language and symbols 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.

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 proprietary programs is regarded as a trade secret, the revelation of which supposedly has disastrous economic effects for its corporate creator.

But there is an alternative: to distribute software *with* its source code. This is the guiding principle of free and open source software (FOSS). At various points in the history of software development, in particular communities of programmers and enthusiasts, and among some 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 send the modifications back to the originator for incorporation in future versions of the software. The core distinction between FOSS and proprietary software is that FOSS makes available to its users the knowledge and innovation contributed by the creator(s) of the software, in the form of the created source code. This not only permits but also encourages interested programmers to become involved with the ongoing development of the software, disseminates knowledge about the inner workings of computing artifacts, and sustains autonomy among the community of software users. 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 an eructation of new business practices.

The terms *free software* and *open source software* are nearly synonymous terms for a particular approach to developing and distributing software. We use the phrase *free and open source software*, or FOSS, to include both notions explicitly. There are important distinctions to be made, however, between the open source and free software movements; we will refer to them

individually when the difference is crucial.

* * *

The FOSS phenomenon is the subject of numerous political, economic, and sociological studies, all reacting to the potential for radical change it embodies. These studies focus mainly on four claims.

First, FOSS is a novel technology for producing software: it "represent[s] a new mode of production--commons-based peer production" (Benkler 2002) and is "a critique of existing laws, contracts, and business practices . . . [with] the potential to explicitly change the 'political-economic structure of society'" (Kelty 2002). Therefore, it is supported by new microeconomic, political, and personal dynamics that may shed light on other areas of economic productivity and modes of collaboration. This new mode of production serves as the basis for examinations of its historical antecedents, parallels from other (sub)cultures, and potential application to other domains of inquiry and cultural and scientific production (Ghosh 2005). The novelty of FOSS, for these investigations, is that it contrasts with the economies of exclusionary property relations, supported by weighty legal structures, that characterized the pre-existing software industry. From the perspective of software engineering, FOSS's proponents tout the superiority of its bazaar-like development model over the rigid cathedrals of proprietary software houses (Raymond 2000). Economists, in turn, are concerned with how this method of production functions, examining the personal motivations and microeconomics of its workforce (Lerner and Tirole 2001), and political scientists investigate the governance schemes that support successful FOSS projects (Weber 2004). Inevitably, some of these claims of novelty are also the subject of critique (Fitzgerald 2005, ; Glass 2005, ; Rusovan, Lawford, and Parnas 2005).

Second, FOSS provides a social good that proprietary software cannot; for example, FOSS may be the only viable source of software in developing nations, where programming talent is abundant but prices for proprietary software licenses are prohibitive. Countries such as China and India have seen in FOSS an opportunity to draw upon their wealth of programming talent to provide the technological infrastructure for their rapidly expanding economies. Microsoft's substantial investments in Indian education initiatives may be prompted by worries that free software might fill indigenous needs instead (Chandrasekhar 2002). FOSS has been cited by Venezuelan President Hugo Chavez as a key element of achieving economic independence from the global North (Leonard 2006). At the 2005 World Social Forum in Porto Allegre, the Youth

Camp focused largely on FOSS issues (Juris 2005). This enthusiasm for FOSS extends to the industrialized First World as well, as many members of the European Union adopt it for governmental administration (Europa 2003).

Third, FOSS challenges many central concepts of intellectual property. Its novel copyright licensing schemes have prompted much debate about the foundations, both ethical and economic, of apparently well-established notions such as property and ownership (Dixon 2003, ; St. Laurent 2004). The emphasis on continual innovation--hailed as the key to FOSS's superior software engineering--puts it into direct conflict with the ideologies of patenting. FOSS forces debate on the distinction between ideas and their expressions that is fundamental to patent and copyright law (Davis et al. 1996, ; Swann and Turner 2004). Indeed, a new cottage industry of legal analysis and application has sprung up to deal with the questions evoked by FOSS's licensing schemes and its opposition to software patents. This is in no small part driven by corporate concern about whether FOSS can coexist with existing business practices.

Finally, FOSS is a threat to the corporate status quo. This facet of FOSS has been trumpeted vigorously by open source advocates, 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 proprietarily-licensed source code produced and sold today (Dibona, Cooper, and Stone 2005, ; DiBona, Ockman, and Stone 1999, ; Raymond 2000). Such advocacy reflects a broader optimism about the ability of FOSS, with other novel modes of industrial organization, to subvert dominant industrial structures. Stakeholders in the status quo are demonstrably aware of this threat, as the leaked "Halloween Papers," revealing Microsoft's sense of the threat of free software, dramatically show (Raymod 1998). To most outsiders, the FOSS community seems remarkably hostile to proprietary software giants. But this adversarial position is fragmented: while some developers indeed hope fervently for the downfall of Microsoft, many seek only for it to show us the code.

* * *

In this book we take free software to be a liberatory enterprise in several dimensions. While the freedom to inspect source code is most commonly associated with FOSS, of more interest to us are the political, artistic, and scientific freedoms it brings in its wake. The title of this book reflects this promise: in a world that is increasingly encoded, our free software carries much potential for liberation. Granted, claims about technology and freedom are nothing new; much of

the early hype about the Internet was rhetoric of this kind. But what is important about the recurrence of such hyperbolic enthusiasm is that it is clearly articulated evidence of a desire for technology to live up to its potential as a liberatory force.

With this book, the investigation of free software becomes broader than those conducted by lawyers, economists, businessmen, and cultural theorists: FOSS carries many philosophical implications that must be carefully explored and explicated. FOSS, most importantly, focuses attention on that often misunderstood creature: software. To understand it as mere machine instructions, to ignore its creative potential and it power to enforce political and social control, is to indulge in a problematic blindness.

We do not contend that "knowledge [or information] just wants to be free," that this is a *fait accompli*. But we do want to understand what this freedom might be and how we might go about achieving it. While the potential of free software is often alluded to, it is not fully understood. This book is partly an expression of a utopian hope, partly the expression of the fear that a liberatory moment is slipping away.

* * *

In the first chapter, we begin with a history of software development as an industrial process, characterizing the emergence of the GNU free software project in the 1980s as a natural step in the evolution of software, one that challenged the anomalous proprietary software regimes that had taken hold in the late 1970s. We show how this history reflects the value of cooperative work, and track the slow move towards the eventual commodification of software; while free software is legitimately regarded as a radical intervention in the software industry, it may also be thought of as a return to software's roots. We investigate the political economies of software, examining the extent to which FOSS invokes or revises traditional notions of property and production. In our narrative, the 1998 schism between the free software and open source movements--where a faction within the free software community changed tactics and language to court commercial interests--is a crucial event. We examine the potential for co-optation that the open source movement has created, drawing it apart from the free software moment, which remains committed to an anti-technocratic, emancipatory, yet pragmatic vision.

In Chapter Two, we examine the ethical positions enshrined in the constitutive documents of the free and open source movements. The freedoms enumerated in the Free Software Definition and Open Source Definition provide normative ethical guidance to the FOSS community.

Building on these definitions, free software licensing schemes grant a suite of rights and freedoms, to programmers and users alike, that are much broader than those granted by proprietary software licensing schemes. At a finer granularity, the particular licenses take different approaches to protecting these rights and freedoms; hence, choosing a license is an important responsibility of the free software developer. The discussion in this chapter is intended both to support this decision-making and to assess the ethical implications of the rhetorical character of these documents.

Chapter Three addresses the FOSS creative process, which facilitates group collaboration and innovation through unique organizational structures. This examination relies on an analysis of the aesthetics of software in general, invoking notions of beauty in science, and uncovering the meaning of "beautiful code" through the testimonials of programmers themselves. FOSS's social and technical organization carries the greatest potential to produce such beautiful code through its affordances for programmers' artistic freedoms.

Building on the previous chapter, in Chapter Four we argue that both the creative possibilities and the scientific objectivity of computing are compromised by proprietary software and the closed standards it proliferates. One necessary condition for the objectivity of science is that its practices and products remain public, open for mutual critique within the scientific community. But the industrialization of computer science and the application of intellectual "property" that is its natural consequence prevent the public practice of computer science. Defenses of commercial computer science on grounds of economic expediency ring hollow when we realize that these have never distinguished science from pseudo-science. We speculate on the shape of computing practices in a world in which all code is free, and show that it leads to a radically different conception of computing as a science.

In the final chapter, we address the role of free software in a world infused with code. In this world, distinctions between human and machine evanesce: personal and social freedoms in this domain are precisely the freedoms granted or restricted by software. In the cyborg world, software will retain its regulatory role, but it will also become a language of interaction with our extended cyborg selves. We point out that questions about the cyborg world's polity are essential questions of technology; the language of the cyborg world must be free, as natural languages are, in order to protect the liberties of our cyborg selves. We argue for transparency in governmentality in the information age and show that free software embodies the anarchist ideal

of eliminating the indiscriminate, opaque application of power.