The Moral Field of Computing

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Jon Smajda

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Joseph Gerteis

December, 2011

Acknowledgements

During the course of planning, researching, and writing this dissertation, there have been many moments where I felt like throwing the whole thing away. Fortunately, I had several people in my life who are always a phone call or office visit away with the ability to talk me down, cheer me up, and offer the advice or reassureances I needed in these crucial moments. My dissertation advisor, Joe Gerteis, offered the perfect mix of encouragement and questioning, pushing me to approach my project from the right point of view and do work of which I could be proud. The members of my committee—Josh Page, Kathy Hull, Ron Aminzade—offered the expertise to help me formulate the right project in the planning stages, and the patience to help me fight through the long, slow process of research and writing. The support of my graduate dissertation support group, consisting of Arturo Baiocchi and Amelia Corl, was vital in providing practical guidance and in keeping my head in the project during the past few years that I've been away from Minnesota. Last but not least, of course, I owe an enormous debt of gratitude to Teresa, Chloe, and our families. The early morning and weekend writing sessions, and the trips back and forth between Minnesota and Kansas, have had me asking for your help and patience all too often during this process, but you have always come through for me. Thank you.

Dedication

For Teresa and Chloe.

Abstract

This dissertation examines the culture of open source software development and debates around "openness" in computing through the lens of sociology. Drawing on contemporary theory and research in cultural, economic, and political sociology, I develop a framework—the moral field of computing—for making sense of the role that group boundaries and moral beliefs play in the day-to-day work of software development. I first show how this field emerged over time during the mid- to late-20th century, and then I show its structure animates the contentious debates and decisions within computing today by analyzing data collected as a participant-observer in several open source communities. For researchers studying computing, this dissertation places the unique culture of software development into a larger context of modern liberalism and sociological research and theory on the relationship between work, democracy, and the market. For sociologists, this dissertation represents a theoretical attempt to understand the relationship between group boundaries, community identities, and moral worldviews through examining an empirical case that has been understudied and undertheorized within the context of cultural sociology and sociological theory.

Contents

List of Tables

List of Figures

Part I

Morality, Culture, Code

Chapter 1

Introduction

In a little over a half century, computers have gone from expensive, slow, room-sized machines accessible to only a handful of privileged elites to a pervasive part of our lives, not only through our desktop and laptop computers, but through our phones, televisions, automobiles, and more. Acknowledging this is basically a cliché at this point. Discussing cultural and moral significance of this change is also not particularly unique. One does not have to look hard these days to find moral panics over the impact of the internet on the social skills of our youth, angry diatribes about how technology facilitates the outsourcing of large swaths of the economy, or, at the same time, utopian bromides on the revolutionary capability of technology to erase all social inequality and difference and usher in a golden age of democracy. In short, we put a lot of moral energy into making sense of computing and its impact on us.

However, in these stories the causal arrows tend to flow one way: from technology to the larger society. This seems to imply that, whatever the impact of technology, the march of technological progress itself follows its own detached, amoral beat. The development of computing, however, has been infused with moral and political deliberation from the beginning. The moral quandaries about computing we face today are not unanticipated: from the earliest programmers, computing culture has a history of grappling with the meaning of working with code and the potential social impact of software. Working with code requires technical skill and attention, but the decisions software developers face involve weighing how communities of people—users, businesses, and other programmers alike—will receive their work and interact with it, and these decisions have complicated and often controversial legal, social, and moral implications. Watching software developers work, one does not find an emotionally detached, technocratic enterprise. Rather, we see a community of morally invested actors doing something they care passionately about, and this passion is grounded in a rich and robust culture of which morality is a constitutive element.

This aspect of morality and computing is less obvious to general bystanders and commentators on the fate of society and technology. However, knowledge that technology did not emerge *sui generis* as a cold, impersonal thing, but rather through a process of moral deliberation and social interaction whose influence is seen in today's products provides an important backdrop to making sense of society and technology today. This dissertation will provide this backdrop and show how the cultural framework developed during the mid- to late-20th century—what I'll call the moral field of computing—continues to animate action within computing today. This framework will be of interest to scholars of technology, of course, but I will argue there are lessons for scholars of culture, morality, and politics, in general, as well. As our work, socialization, recreation, and education are increasingly mediated by computers and the internet, the issues surrounding the morality of software development and distribution will become more and more mainstream. The moral panics alluded to above may be premature in their sensationalism and simplicity, but they are correct that the technology developed in the second half of the twentieth century has the potential to radically transform our society. Software is, as Netscape co-founder Marc Andreesen put it, "eating the world": one industry after another is being turned upside down by the internet and the software programming tools that run the internet: retail, entertainment, publishing, telecommunications, finance, energy, education, and health care.[1] The relevance of understanding the social and cultural conditions in which this technology is developed goes beyond a niche subdiscipline of those interested in technology—just as an understanding of the industrial revolution had field-wide relevance at the founding of sociology as a discipline.

1.1 The Moral Field of Computing

My account of the "moral field of computing" emerged out of two parallel research tracks. First, I offer a theoretical re-analysis of secondary sources telling the story—or, to use Somer's (1994) phrase, "conceptual narrative" (see page 16)—of computing's development in the second half of the twentieth century (Part II). Second, I undertake an analysis of discourse in four

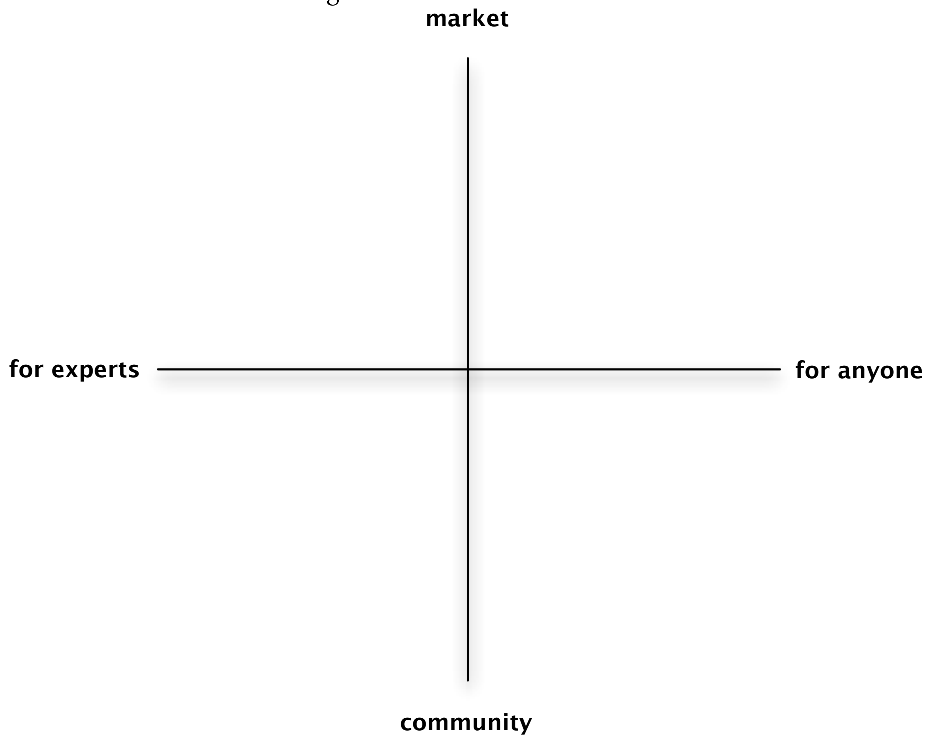
[1]http://online.wsj.com/article/SB10001424053111903480904576512250915629460.html

contemporary case studies in the domain of free and open source computing (Part III). My historical reading and interpretation went hand-in-hand with my focus on contemporary cases: the patterns I saw developing in the 1960s and 70s continued to shape the conflicts I was witnessing in the 2000s. These patterns involved boundaries and boundary-making processes about communities, experts, and markets, as well as vigorous deliberation over the nature of cooperation, public goods, and exchange. To make sense of these patterns, I developed a framework I call the "moral field of computing."

The moral field of computing asserts that the culture of computing is shaped by two core dimensions of morality and identity. First, the moral field is shaped by tension between visions of markets and communities. When participating in the development of software, is one participating in a *community* building a *commons*? Or is one participating in a *market* thet builds *products*? In particular, I focus on the phenomenon of "free and open source software," where the source code to one's software is distributed and users of the software are free to modify the source code and, under certain conditions, redistribute their modified versions of the software. "Open source" has become a buzzword for evoking transparency and collaboration. In practice, the development of free and open source software is embedded within networks of developers and businesses with competing, often contradictory, definitions of fundamentally moral concepts like "openness," "fairness," and "freedom." For some advocates, open source is a means to carve out a space for developers to collaborate independent of business influence and competition, while for others, open source is an inevitable victor in the market for software because of the gains in efficiency it offers. In both cases, "open source" is an inherently *liberal* discourse (Coleman and Golub 2008), grounded in the core tensions over markets, community, and freedom that are constitutive of modern liberalism.

Ideological tension between "markets" and "communities" is not sufficient to understand the moral culture of computing, however. Beneath the discursive surface is an enterprise undertaken by *people*, who organize themselves into groups and make sense of their own position vis-à-vis others. Software development is a technical enterprise involving skill and expertise, yet there are two populist strands within computing culture that focus on preserving an exclusive culture of elite hackers, on the one hand, or building technology that will empower the larger society and world, on the other. For instance, in chapter 7.2 we will see how the goal of building a "Linux for Human Beings" moralizes many otherwise technical decisions about the design of the Ubuntu Linux distribution. These conflicts are, in part, shaped by

Figure 1.1: The Moral Field



competing demands of building an operating system that is both a product for the mass market as well as public good built by a community (our first dimension), but these definitions of group *mission* are complicated by conflicts over group *boundaries*. As we see debates over apparently technical issues become heated "flame wars" (Dery 1995), it's not just a matter of stubborn geeks arguing over technical matters, but moral debates over who who counts as "the community," and whose needs, preferences, tastes, and skills matter.

The realization that the questions of group boundaries and moral visions are intimately related leads me to conceptualize the culture of computing as a *field* with one axis representing competing conceptions of identity and community (a community of experts vs. a community extending to all of humanity), and the other representing competing moral and ethical visions (an ethic of community vs. an ethic of the market). (Figure 2.1)

My choice of "field" borrows from the work of Pierre Bourdieu (1983, 1992), for whom field is a relational space within which agents stake out positions in relation to one another, with shared, tacit "rules of the field" governing individual beliefs, motives, and assumptions. The concept of field is particularly fitting for several reasons. First, the spatial metaphor: it's not just that there are two kinds of questions—about group boundaries, about moral visions of markets and communities—that one can observe animating debate in computing culture, but that these appear to be linked together into specific configurations of how to situate individuals and actors onto a structured, cultural terrain. Second, I argue that this structure is internalized by individuals as a moral and political mapping of actors and events in computing. Participants in the computing field—hackers, corporations, end-users, and so on—position themselves and others on a moral terrain built upon attitudes towards what kind of community is worth building and who counts as a part of that community, and what kind of moral values should guide interaction in that community.

1.2 Roadmap

This dissertation will proceed in three parts:

First, this section provides an overview of the literature, theory, and methods behind this project. I will situate my account of the moral field within the sociological literatures on morality, culture, and markets, as well as the broader social scientific literature on computing, and "open source" computing, in particular. There are several strands of social scientific research on open source from the perspective of economics, law, and organizations, but none of these adequately address the distinct political culture I will describe in the free and open source software world. At the same time, several social movements—such as the free culture movements—have drawn heavy inspiration from free and open source software. I will describe how these movements have attracted some intellectual attention even though the culture of their source domain—computing—has remained undertheorized.

Next, I provide a chapter on research methods that describes the methodology employed in the course of this dissertation. I made several methodological choices based both on pragmatic but also theoretical grounds during the course of this project, and I discuss these decisions and the advantages and trade-offs they entail. I utilize discourse analysis of online discourse from publicly available sources such as blogs, mailing lists, and forums. Additionally, I argue that the work represented by this dissertation sets the stage for future research, and research methods, such as in-depth interviews or more technical discourse analysis of specific data sources such as project bug trackers.

In Part II, I will show how the moral field of computing emerged over time. First, I review the history of computing from the 1950s to the 1980s. In the early days, only a handful of tight-knit expert communities had access to computers, but a strong identity and ideology was cultivated within these communities. In the 70s, computing began to spread: to more and more universities as well as to the general public. As hardware prices plummeted, a growing group of "home hackers" thrived as did businesses selling hardware and software to them. By the 90s, however, computers—and the internet—were everywhere: transforming workplaces and homes alike. In other words, what began as a culture for small expert communities had gone mainstream and commercial. In response, hacker purists worked to institutionalize their traditions in the form of the free software movement, but this movement fragmented by the mid-90s. By this point, we will see the moral field in full play: competing ideologies of markets and democracy, and competing definitions about community boundaries and priorities clash, moralizing the process of software production and distribution.

In Part III, I will use current case studies to show the moral field in action. I divide this part into two chapters, focusing on the relationship between licensing and usability, respectively, to the discourse of "open" and "free" in computing. The chapter on licensing will compare debates over the two forms of open source licenses—the "liberal" and "viral"—in the context of two open source projects: Android, a mobile operating system, and WordPress, a web application for content management. In this chapter we see how legal decisions about licensing software are made and publicized in terms of the moral value of markets and communities. Next, a chapter on usability looks at the deliberation over usability and openness in the context of two companies: Apple and Canonical, the company behind Ubuntu Linux. If end-users are not programmers, is the more "open" approach one which is less hacker-friendly but more "user-friendly"? Does this choice have to be made? Each of these case studies show communities of designers and developers alike wrestling with these questions. Behind these ideological questions, however, lie tensions between non-profit community projects and for-profit businesses, and between groups of people who have strikingly different views about what makes good software and whose opinion should matter when creating it.

Chapter 2

Social Science and Technology

Chapter Preview: Open source software has received much scholarly attention from scholars of business, organizations, and law, but computing culture, and open source in particular, has thus far been understudied within sociology. However, the literatures on the intersections between culture, morality, politics, and markets have much to offer the subject. In particular, the study of morality is enjoying a renaissance among sociologists, particularly among cultural sociologists and economic sociologists. Additionally, a focus on the processes through which social and symbolic boundaries are created and maintained has been another growth area in sociological theory and research in recent years. Those studying the culture of technology and software development can enhance their understanding by contextualizing their work within these sociological fields. Additionally, sociologists interested in culture and morality have much to gain by paying attention to this interesting, ascendent subculture. In this chapter, I review existing research on computing and open source as well is research on culture and morality in sociology, and I show how this sociological work has informed the approach I have taken in this dissertation.

Much of the existing social scientific research on technology and computing can fit under the umbrella of "Communications and Society": research on the social impact of technology, the internet, and the "knowledge economy" (Powell and Snellman 2004). This general topic has spawned "Grand Theory"-style work by academics such Manuel Castells (1996, 2001), Benkler (2006), and Wellman (2001) as well as more popular works such as Shirky (2008), Surowiecki (2005), and Tapscott and Caston (1993), Tapscott and Williams (2006). More targeted research has been done on a handful of topics. For instance, there has been much research on the "digital divide"—inequality and access to computers and the internet (DiMaggio et al. 2001, Guillen and Suarez 2005, Keller 1996, Hoffman et al. 2000, Martin and Robinson 2007, Ono and Zavodny 2008). More recently, the rise of "social media" and "social networking" sites such as Facebook has led to research on how people are using these sites (Hargittai and Hinnant 2008, boyd and Hargittai 2010, Lewis et al. 2008) as well as research on how "digital literacy," or lack thereof, shapes how people, and young people in particular, make sense of what they find online (Daniels 2009, Hargittai 2010).

On the surface, this research has much in common with mine, and sociologists are well represented in this group. However, my focus is quite different. While these researchers are interested in the *impact* of technology, I am interested in the *production* of technology, and software development in particular. The social impact of technology will be a common topic in my work here, but not as an empirical question in itself. Rather, I am studying how the people who produce technology take their beliefs about the impact of technology into consideration in the production of technology.

With that in mind, there are two other bodies of social scientific research relevant to my work here. First, scholars have attempted to make sense of how free and open source software works from an organizational, legal, and economic perspective. Second, a wide range of "open source"-inspired movements have received attention. However, sociological perspectives are not as well represented in these arenas. While some work on the culture of open source exists, most of these have not been from an academic perspective, but written by journalists or insiders working within open source communities. Consequently, sociologists working on culture, politics, and morality have not, by and large, been exposed to a fascinating domain that offers them much to chew on theoretically and empirically. Additionally, those works that do focus on culture and computing have lacked the larger context that a sociological analysis can offer, situating what they see in software development with what sociologists know about things like social and symbolic boundaries, moral discourse, identity, and movements.

In this chapter, I will provide an overview of two sets of literature. First, the existing research on technology and open source outside of sociology I alluded to above. My review here will not be comprehensive, but will serve to illustrate that a) people are paying attention to this topic, but b) they are all still quite different from what I will be engaging in. Second, I will turn towards sociology and review two strands of theoretical work and empirical research in contemporary sociology. First, I look at research on culture and morality, particularly as these intersect with markets and politics. Second, I turn to research on boundaries and boundary-making processes. Finally, I show how these literatures informed my theoretical framework in this paper, and suggest what this dissertation has to offer the field.

2.1 Existing Research on Technology and Open Source

There are basically two big groups of existing research on open source and software production from a social scientific perspective. First, a large body of work approaches free and open source software as an economic, organizational, and legal curiosity. Why would a programmer give their work away for free? How can such a decentralized form of organization produce great software? How do licenses such as the GPL fit within the traditional framework of copyright and intellectual property? A second group of work looks at phenomenon that are what I'll call "Open Source by Metaphor," non-programming domains that have adopted the language of "open source" or "openness" to describe new practices.

2.1.1 An Economic, Organizational, and Legal Curiosity

The issue of motivation has recieved much attention from economists (David and Shapiro 2008, Krishnamurthy 2006, Lerner and Tirole 2005b). On the surface, thriving open source projects seem to challenge the basic assumptions economists make. As Lerner and Tirole (2005a:48) ask, "Why should thousands of top-notch programmers contribute freely to the provision of a public good?" Many of their answers will be unsurprising to an audience of sociologists—programmers enjoy programming so open source projects are fun, and contributing to open source projects is a way to gain status within the community. Some of the answers require a little more of an understanding of how software development works. Most programming that is done is neither open source or proprietary: it is never distributed at all. Most programming is done in a local context, building one-off solutions for specific companies or organizations. Shortcomings or bugs in the tools used to build this software get in the way of work the programmer *is* being paid for, therefore it is in their interest to chip in and improve the software when they can. Open source software is appealing for this very reason: they can improve the software themselves rather than filing a bug report with a large corporation and hoping they pay attention.

In other words, open source software is driven by what Eric von Hippel (2005a, 2005b) calls "User Innovation Networks." From an economic perspective, willingness of businesses and individuals to freely share individual improvements instead of hoarding them follows the logic of precompetitive collaboration (Weber 2004:21-22). It's prohibitively expensive for any one organization to build an entire stack of software internally. For example, say you're a company that provides networking services for clients. While it may help your competitors to have access to the performance improvements you've made to, say, the Python programming language, a) you're not really in the programming language business, b) you will also benefit from improvements your competitors have made to, maybe, the Apache web server, freeing both of you up to focus on where you do compete. It's akin to how construction companies don't mind collaborating to adopt common standards when it comes to parts and equipment: it lowers the cost of materials for all of them so they can focus on competing where differentiation actually matters.

In the internet age, intellectual property and copyright are hot topics, with the internet transforming the business models of content industries in news, entertainment, and education. In this context, the alternate vision of copyright presented by free and open source software has gained the attention of activists (see below) as well as legal scholars (Grimmelmann 2005, McGowan 2005, Seltzer 2006).

2.1.2 Open Source by Metaphor

Open source software achieved notoriety in the mid-1990s during the "dot-com bubble," not only within the technology industry but also among media pundits and technologists, and the popular appeal of the "open source" label has remained. In fact, one can find activists, politicians, businesses and commentators pushing for the "open sourcing" of just about every possible domain of social life. In government, the Obama administration has caught the "open" meme. On election night in 2008 on CNN, Alex Castellanos lauded Obama as an "open source president," citing hacker Eric Raymond's book on open source software, *The Cathedral and the Bazaar* (Raymond 2001). Whitehouse.gov now runs on the Drupal open source content management system and Obama created a new Chief Technology Officer tasked with implementing an Open Government Directive to bring "transparency, participation and collaboration" to the White House. Before Obama, Howard Dean's 2004 campaign pioneered the label of the "open source campaign," with campaign staffers drawing on Linux, in particular, as a role model for running a collaborative, decentralized campaign (Kreiss 2011).

The White House isn't alone: the Committe on Homeland Security held a hearing on "Using Open-Source Information Effectively" (Congress 2005). In 2007, former United States Air Force pilot John Robb wrote a best-selling book classifying the Iraqi insurgency as "open source warfare" (Robb 2007). Inspired by Washington's newfound receptivity to all things "open," 70 major tech companies (including Google, Novell, Mozilla, and Sun) and the Electronic Frontiers Foundation launched "Open Source for America", which aims not only to "raise awareness in the U.S. Federal Government about the benefits of open source software" but to encourage the "incorporation of open source community dynamics to enable transparency" throughout government. In virtually every other sphere of the social world, one can also find calls to open source: from open source biology (Hessel 1999) and medicine (Woodford 2004), to open source urban planning (Buskirk 2009) and disaster relief (Hodge 2010, Sahana Software Foundation 2010).

The Free Software wing of the "free and open source" movement have also proven influential in a "free culture" movement that has emerged in the context of online "mashup culture" and the increasingly draconian intellectual property policies pursued by the content industries. The Creative Commons license, for instance, was designed to serve as the GNU GPL of art and literature: preserving freedoms for consumers of content and providing minimally restrictive licensing options for authors. These movements have extracted an "action repertoire" from their role model of open source software (Breindl 2011), utilizing online technologies to facilitate decentralized, geographically dispersed activism, and encourage cultural innovation and creativity through "modulations" (Kelty 2008). These metaphoric "open source" movements have, oddly, received more high profile attention from social scientific academics (Lessig 2005, 2008, Kelty 2008) than has the moral and cultural dimensions of computer programming specifically, as I am here. Perhaps this is because the political and moral project is much more clear when it comes to *culture* than "code." Perhaps it is because free culture advocotes frequently view their efforts as activism, while programmers tend to point to the "intrinsic motivation" of open source's utility (Lakhani and Wolf 2005).

2.2 Bringing the Sociology In

While these existing approaches to understanding free and open source software comprise interesting contributions to their fields, as a sociologist, I found myself gravitating towards a different set of issues. In particular, the distinctively moral nature of a discourse about "free" and "open" was striking to me, as well as the ways in which "community" was mobilized to describe participants in open source.

There has been some work in this direction, particularly within anthropology by Gabriella Coleman (2004, 2004, 2008, 2010). In a series of papers, Coleman has articulated several important features of hacker culture that are, for the most part, consistent with my account here. Coleman has emphasized that hacker discourse and its focus on defining and enhancing freedom is enmeshed in liberalism (2008) and a form of "political agnosticism" that simultaneously engages in vibrant moral deliberation while stridently maintaining that such deliberation is "not political" because it aims to "enact a *universal sphere* for the flourishing of free forms of action and thought" (2004). Like my account here, Coleman asserts that there is important cultural diversity among hackers, pointing to three "moral genres": "Crypto-freedom," "Free software and the politics of inversion," and "The underground."[1]

My approach is more broad in some respects and more focused in others. The vast majority of this dissertation will focus on Coleman's second moral genre, concerning free and open source software, and the diversity *within* this "genre." In this sense my focus is more narrow, but in another sense my focus is more broad: I am not just concerned with "hacker culture" but with the discourse of "open" across the entire domain of technology and software development. Additionally, while I will say much about language and discourse, my explanatory framework goes beyond language: social boundaries and social structure are the crucial ingredients in giving shape to the moral field. Liberalism does, indeed, weave in and out of conflicting moral positions taken in this field, but it's not necessarily *liberalism* doing the shaping here. In fact, liberalism as a cultural discourse may thrive precisely because it is able to be deployed by all opponents in these core debates, functioning as a shared language for negotiating conflicts born out of different social positions and relationships to the market.

These differences are in part simply due to choices about project scope and what to study, but I believe they also represent differences in disciplinary focus. In particular, two streams of research within sociology have informed my perspective and interpretation of events: first, work within the sociology of culture on morality and, in particular, morality in the context of political culture and economic culture; second, a larger theoretical framework asserting the

[1]"Crypto-freedom" concerns pro-cryptography advocates and the "crypto-wars" (Kahn 1996) while "the underground" refers to software "piracy" and file-sharing, each of which are peripheral movements in the context of this project.

importance of social and symbolic boundaries in understanding human social behaviour.

2.2.1 Morality and Culture, Markets and Democracy

The first body of literature concerns culture and morality, and in particular, morality and its role in both markets and politics. While morality is not a new interest of sociology (Abend 2010), the past decade has seen a revival of interest in the sociology of morality (Hitlin and Vaisey 2010, Winchester and Hitlin 2010) Collectively, these works present a perspective on social life with morality at the center of how people structure their social worlds, interact with one another, evaluate the actions of others, and construct self-identities.

Labels such as "free software" and "open source software" are laden with moral connotations about the value of collaboration, sharing, and the common good. And yet, open source software is big business, with many of the most profitable companies in computing—Google, IBM, Oracle, Apple, even Microsoft—hosting or sponsoring the development of open source projects. The intersection between the profitable, proprietary services and software produced by these companies and the open source commons is a fertile ground for the examination of the tensions between different moral visions of communities and markets. Understanding the moral visions that support how people understand markets has received much attention in sociology in recent years (Fourcade and Healy 2007, Lamont 1994, Massengill and Reynolds 2010, Wherry 2010, Zelizer 1979).

With the rapid growth of the information technology industry in the past half-century, software development occurs at the intersection of apparently conflicting models of moral action. On the one hand, free and open source software is produced by a community of programmers, both professionals and hobbyists, and this code is a shared public good. On the other hand, software—open source and proprietary alike—is an industry that is thriving, profitable, and powerful. Clearly, on a certain level, free and open source software and for-profit businesses can, and do, co-exist: the success of businesses that continue to fund open source software development attests to this. But what sort of moral system supports such a bifurcated system?

Setting the boundaries between market and non-market action has long been a morally contentious process. While markets are typically associated with rational, self-interested, amoral action, in fact a great deal of moral negotiation occurs to both facilitate and challenge markets. Two dominant moral visions of markets—what Fourcade and Healy (2007) call "The

Liberal Dream" and "The Commodified Nightmare"—view markets as either a positive vehicle for nourishing honesty, civility, and cooperation, or as having a corrosive effect on these values, respectively. Defining the relationship between the market and objects and relationships requires navigating the boundaries of sacred and profane drawn by each of these visions. For instance, Zelizer (1979) illustrates the process of "sacralization" through which life insurance—once considered a taboo means of profiting from death—became a morally acceptable, even morally responsible, commodity for sale in the market. In the opposite direction, Zelizer (1985) shows how children were removed from the labor market as child labor came to be viewed as a violation of sacred childhood. Similarly, Healy (2006) examines how both opponents and proponents of the market exchange of donor organs appeal to morality: opponents classify human organs as sacred objects that have no place in the market, while proponents sacralize organ donation as an altruistic act.

Another angle on the relationship between markets and morality is that programming is *work*. While it is true that much open source project participation is volunteer-driven, larger projects in particular are seeing increasing levels of professionalization. For example, Linux kernel contributor Jonathon Corbet analyzed code contributions to Linux between 2008 and 2010 and found that 75% of the code contributed to Linux is now written by developers paid to contribute to Linux.[2] (Top employers of these contributors: Red Hat (12%), Intel (8%), IBM (6%), Novell (6%), and Oracle (3%).) With this in mind, open source software can be interpreted as a current example of a long line of efforts for workers to increase control over the fruits of their labor through a kind of participatory democracy. From Carol Pateman's work (1976) on democratic management techniques and worker self-management in the 70s, to worker cooperatives in the 80s (Rothschild and Whitt 1986, Cheney 2002) and "Real Utopias" in the 2000s (Fung and Wright 2003), research on efforts to democratize work has a long tradition in sociology. Open source software offers an interesting variation on this theme: egalitarian, participatory practices that began in volunteer communities and universities are now being funded and supported by corporations, who have found that it's in their interest to have a semi-autonomous, relatively democratic commons of free software to draw on. As we'll see in later chapters, the relationship between these complimentary modes of software production is sometimes peaceful, but often full of tensions.

A useful way to think of the relationship between the specific case of software development

[2]Source: http://apcmag.com/linux-now-75-corporate.htm

and these larger societal-level discourses is offered by Somers (1994). Somers argues narrative is an "ontological condition of social life" and that people locate themselves within "a repertoire of emplotted stories" (p. 614). In addition to the narratives people construct about their own lives ("ontological narratives"), Somers makes the distinction between "public narratives"—those attached to institutions such as church, state, family, or, say, open source software projects—and "metanarratives" or "masternarratives" that are the "epic dramas of our time": progress, decadence, capitalism vs. communism, the individual vs. society, etc. One way to describe my work here is linking up the "public narratives" that software developers use to make sense of the institutions they are operating within, and the "metanarratives" of contemporary liberalism. In making these connections, I resort to another form of narrative identified by Somers—"conceptual narrative": narratives used by social researchers that draw on social forces to explain and understand the world. Among these "social forces" I find the dynamics of *boundary processes* and group identities to be particularly important.

2.2.2 Identity and Boundaries

The moral negotiation over the relationship between open source communities and work, and between workers, volunteers, and markets, does not occur in a vacuum of abstract reason. It is tightly connected to the social position participants find themselves in and how social and symbolic boundaries (Lamont and Molnár 2002) are drawn around different practices and groups of people.

Among social psychologists, there is a growing consensus that moral principles are subservient to moral identity (Blasi 1984, Stets 2010). Identities, in this view, are not static, essential features of an individual or group, but are *performative* and *situational*, with *group boundaries* and processes of *boundary making* as central features of the social world. The goal is not to neatly categorize every act and actor into a typology of logically consistent moral worldviews, but rather to recognize that moral beliefs, moral identities, and the moral boundaries demarcating the relationships between beliefs, identities, and actions, are crucial ingredients in how people interpret and interact with the world around them. Boundary-making, particlurly in a heavily textual domain such as online open source communities, is often visible in the narratives communities engage in.

For instance, in the realm of free and open source software, one of the ongoing negotiations in the "emplotted stories" concerns where to draw boundaries around who belongs to "the community": programmers and hackers only? Exclusively open source hackers only or those that work for proprietary software companies that use open source software, too? What about non-programmers who contribute? What about end users and "evangelists" who promote open source software? In short, instead of thinking of "the community" as a clear, obvious thing that exists—the *idea* of the community and the *boundaries* around who and what counts as "in" the community—sociological work on boundaries suggests we focus on how these aspects of community are negotiated and contested in practice.

This approach has a long tradition (i.e. Barth 1969) but has been resurgent in the past decade (Lamont and Molnár 2002). Sociologists have employed the concept of boundaries to understand science (Gieryn 1999), religion (Edgell et al. 2006), ethnicity (Brubaker 2004), sex and race (Nagel 2001), sex and gender movements (Gamson 1997), nationalism (Bail 2008), and class (Bourdieu 1984).

Several simple boundary-making and boundary-enforcing claims could, and have, been made about computing culture, and the relationship between proprietary, open source, and free software in particular:

- Free software advocates are ideologues who want to constrain freedom in the name of freedom, and in doing so prevent their software from reaching the masses they claim to serve.
- Open source software is a business-friendly cop-out that compromises the ideals and principles of free software for short-term profit.
- Proprietary software holds users hostage to the demands of the software vendor by taking away their freedom to control code running on their own machines.
- Open source advocates care a lot about developer freedom but don't seem to care about the freedom of ordinary users, who can't program but who can benefit from software that is easy to use and supported by a real company.
- Requiring software developers to give away their code violates their freedom and is a disincentive to do great work because freeriders can come along and profit from their efforts while doing less work.

In the course of this dissertation, I'll show people making each of these arguments, and more. Sociologically speaking, the question is not whether or not one of these arguments is right, but to step back and try to understand the conditions and processes that led to a field upon which actions are classified in moral terms such as this. The first piece is understanding, following the cultural and economic sociology cited above, that discourse around computing is situated within markets for information technology products and within the modern political culture of liberal democracy and free market economics. In the above examples, we see boundaries being drawn between communities and corporations, between experts and end users, and between "open source idealists" and "market pragmatists." These boundaries shape how actors interpret the motivations of others as well as make decisions about what kind of code is good code. Drawing moral boundaries around software development practices, therefore, involves engaging with more abstract moral questions about freedom, community, and exchange that are both central to and highly contested within our modern moral order.

2.3 The Moral Field

It is with these core concepts and background, then, that I proceeded to "go native armed," as Wacquant (2009) put it. As I'll describe in Chapter 3, over the course of doing this dissertation, I learned how to program, made some relatively modest contributions to some open source projects, including releasing a few of my own, and even worked part-time as a web developer using open source software. I also become a genuine fan of many open source projects and project leaders. My interpretation of the culture and events I encountered, however, were shaped by my background knowledge as a sociologist as I was sensitized to focus on social and symbolic boundaries and the role the moral negotiation of concepts such as "open" and "freedom" play in boundary-making processes.

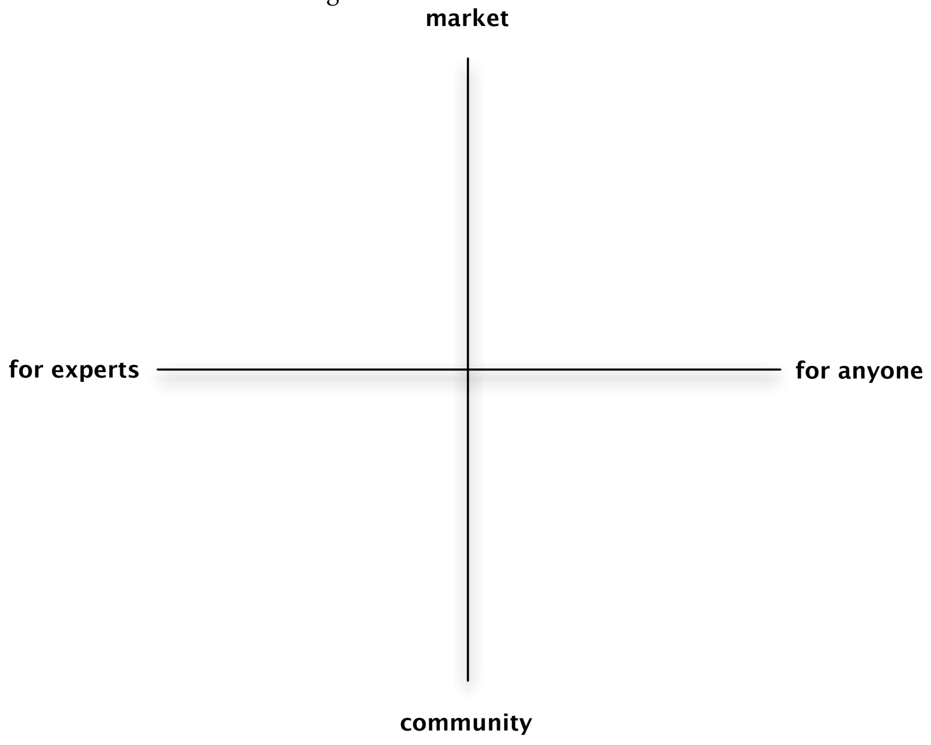
When viewing computing history and current events from this perspective, I found that the social structure of open source computing is shaped by two core dimensions of morality and identity. The first dimension concerns the relationship of software developers to the market, and hinges on the answer to the question, "Are we a *community*, or are we a *market*?" In other words, is the objective to build a *commons* or a *product*? These questions are, in part, grounded in the classical division between visions of the market as either instrument of virtue or distorter of human values, but the story is not that simple.

As we shall see, there are certainly moral entrepreneurs (Becker 1995) drawing lines in the sand, but the tension between "community" and "market" is salient not so much in that it separates mutually exclusive groups of people from one another, but because actors consistently struggle with positioning themselves and others in relation to community and markets. How this proceeds, I argue, reveals a second dimension of morality and identity that hinges on the question of group boundaries: when making decisions about the rules and principles governing our group, who deserves consideration? Are we primarily a group of *experts*, or do loyalties and obligations extend to the *public*, to all of humanity? What is the moral relationship, in other words, between software developers and the public that uses their software?

Together, the answers to these boundary-drawing questions lead to different ideas about how members of the computing field should interact with one another; about what shared goals, rules, norms and values ought to guide interaction. These two dimensions form the two axes of what I will call the *moral field* of computing: competing conceptions of identity and community (a community of experts vs. a community extending to all of humanity) on one axis, and competing moral and ethical visions (an ethic of community vs. an ethic of the market) on the other. (Figure 2.1)

“Open source,” I argue, is best understood within the context of this moral field. The term was chosen, and has succeeded, precisely because it can be adapted to fit the needs and concerns of actors located at multiple points in this field. “Open source” is a “moral vocabulary” (Lowe 2010) that wraps source code in a protective coating from the profane market, preserving the sacred act of code sharing while allowing code to enter the market. However, this is a contentious process. As I illustrate with the moral field, participants in free and open source software have different conceptions of the market and the role that their software should play in the market, as well as different ways of drawing group boundaries that influences their understanding of what makes open source code sacred.

Figure 2.1: The Moral Field



Chapter 3

Research Methods

Chapter Preview: My research is based on three years of immersion in the software development community. I systematically tracked blogs, forums, mailing lists, and public deliberation by participants across many open source software projects, and I also learned how to program and made modest contributions to several open source projects myself. After considering several methodological approaches (such as interviews), I settled on a methodology combining participant observation, ethnography, and discourse analysis in an online environment. I defend this decision on pragmatic grounds (start with what's there) and also in terms of validation: this is a relatively new area of inquiry for sociologists and the kind of broad sketch of computing culture that is a valuable starting point for other research is best served by these methods.

3.1 Research Design

In setting out to study the topic of moral and political culture in software development, I considered several options. First, I considered an interview project. I had conducted in-depth qualitative interviews before (as part of the University of Minnesota's American Mosaic Project), so I knew how to go about writing and conducting such interviews, and I also liked the end product in-depth interviews enabled. However, I decided against this option. For one thing, it was apparent to me that the fundamental problem in approaching computing culture was not a paucity of discourse about moral matters, but an abundance of it. Most open source projects today consist of geographically disperse individuals and communicate mostly online, much of it in public, on project mailing lists, IRC, forums, blogs, twitter, and at conferences (which are frequently posted online). Additionally, one of the characteristics of this community that hooked me to begin with is the tradition of relatively open debate about the moral and political issues of their efforts. Most of the time sociologists set out to study some community and end up asserting that what the community thinks of as "just how we do things" is in fact driven by power, inequality, notions of morality, etc. In this case, however, we find a community already engaged in deliberation, sometimes heated, about these issues. Yet, as I discussed in my literature review, sociologists of culture and morality have not adequately explored the culture of software development. So to do interviews, it seemed, would be getting ahead of myself—just combing through the material that already exists online, for the whole public to view, is a large job. No doubt, there are corners of the problem in-depth interviews could address that I will miss without them, but these interviews will be better if conducted as a future project on the foundation of a study of the material already available.

So how to best capture and interpret the discourse that is out there? I considered several options. My first instinct was more formal: identify a set of sites where deliberation occurs and devise a sampling strategy to pull text into a corpus of data I could then query and analyze in a robust, systematic way. There were two challenges to this approach: 1) choosing a set of sites and sampling strategy to use, and 2) how, technically, to take the data from such diverse sources and organize it into a coherent, uniformly-formatted collection of data. Unsure about the answer to the first question, but knowing I would eventually have to answer the second, I took on this second problem first.

I wrote a Unix bash script that, when passed a URL, would send the URL to Elinks, a text-based web browser, and save the content of the Elinks-formatted page to a simple flat-file database. Elinks strips all images and stylesheets from the site and presents just the plain text of the page. It preserves formatting through the use of line breaks and indentation and converts hyperlinks into a footnote-style system where links are all numbered throughout the page and then the corresponding URLs are listed in a numbered list at the end of the document. This doesn't result in complete uniformity across sites, but it's close. At minimum, everything is now in a simple, plain text format that is easily portable between different methods of analysis.

In addition to parsing and cataloguing the content of these pages, the script prompted me to "tag" each article as I added it. For example, a forum thread discussing the design changes Canonical was making to Ubuntu would be tagged `ubuntu`, `design`, `usability`.

These document-level tags were added at the top of each file, but I could also add line-level tags inside the document as well. I then wrote another bash script that would allow me to search through all saved pages, either by tag or by doing a full-content search of the entire database. (They are not pretty, but I have included both scripts in the Appendix, starting on page 146.)

But I still had not decided on a satisfying sampling strategy. The problem was a mismatch between what sounded like a good strategy on paper, and what I was actually experiencing "in the field." And that last phrase is key: I wasn't just doing "discourse analysis" on some data set, but I was gradually becoming a participant-observer in the world of software development. I learned how to program and made modest contributions to various open source projects—I wrote and released a few plugins for the WordPress blogging platform, for example. Additionally, as I write this, I am employed part-time as a web developer using the very open source tools I began studying as a sociologist. So what began as a "discourse analysis" project gradually morphed into more of an ethnography or participant-observer project, with the "site" for this research being the extensive network of online outlets and venues for deliberation within various open source communities.

At the same time that I was deciding on how to best study more current events in the computing world, I was reading a large number of historical accounts of the development of computing from the 1950s on. While reading these works as a sociologist, I noticed some patterns that authors of these histories had hinted at, but not fully articulated. In short, I developed the basic shape of the Moral Field of Computing during my historical research about computing. At the same time, I was witnessing the legacy of this cultural framework in the day-to-day events I was observing. The development of my theoretical framework, therefore, was an iterative process between building Part II and Part III. Developing an account of how the moral field emerged over time at one moment, and then observing how the structure of the moral field revealed itself in the actions of contemporary actors in software development and computing today.

3.2 Data Sources

As social scientists, we generally prefer to study events that have already happened. The data collection strategies are more clear and complete. But, in my case, current events as they happened were simply too fascinating for me to overlook them. Picking events from 2005 would have been much easier methodologically—and maybe it's just the wannabe journalist in me—but I felt I needed to find a way to bring current events, as they were unfolding, into my analysis. One of the challenges of this is that my "data collection" strategy necessarily became more agile and improvisational. Let me give an example. In the chapter on usability later in this dissertation, I will discuss the advent of the Ubuntu Design Team, the challenges they faced, and the way their interface changes were received by the Ubuntu community. Sources I draw on in covering this include some predictable sources (the Ubuntu Forums) that I would have "marked" ahead of time, but others that I would not have anticipated: a thread in Ubuntu's bug tracker, a new blog started by the Design Team, as well as commentary by others in the computer user interface world that were drawn into the debate. For example, I remember seeing the following quote—posted by Ubuntu founder Mark Shuttleworth in response to a "bug report" critiquing an interface change in Ubuntu—on Hacker News, a social link aggregation site popular among software developers:

> This is not a democracy. Good feedback, good data, are welcome. But we are not voting on design decisions (Shuttleworth 2010).

From there, I found other sites linking to the thread (using a simple "link:" search in Google). In other words, I followed the story much like any other member of the open source community interested in the topic might, or perhaps like a journalist following a beat. Now, to be sure, one could take this incident and build a more formally-stated sampling strategy around it: take discussion threads attached to bug tracker systems used by open source projects and track the frequency and ways in which moral or ideological considerations enter into either classifying something as a bug or into negotiating the acceptable solution to a bug. This dissertation will be full of such suggestions, in fact. As I stated above, I decided that, at this time, a more general project setting the foundation for more targeted research down the line was the best course.

Over time, patterns in what sources I found myself drawing on emerged. In particular, I draw on four kinds of sources: publications, high-impact websites, community websites, and news sources.

3.2.1 Publications as Data

Many of the citations in this dissertation are to be viewed as "publications as data" as much as, if not more than "publications as literature." What I mean by this is that in studying the discourse of computing culture, a natural place to start is in influential, published works that are widely read and cited within the community. Such publications are frequently exemplars of the "double hermeneutic" (Giddens 1987): intended as factual accounts of the culture, the concepts and theories advanced impact the community they describe. The first few sections of this dissertation—which sketch out the history of computing and outline my framework for making sense of the culture that emerged from this history—are particularly heavy with publications in this category, including:

Manifestos: Several works have achieved the status of computing classics and are frequently cited as "must-reads" for newcomers to programming and open source, in particular. These include books such as Eric Raymond's *The Cathedral and the Bazaar* (2001) and *A History of Hackerdom* (2003), several essays by Richard Stallman on free software (Stallman 2001, 1999, 2002), a series of essays by leaders of open source projects published in an edited book series called *Open Sources: Voices from the Open Source Revolution* (1999), as well as official documentation such as the Debian Social Contract (2004), the Open Source Definition (Open Source Initiative 1999) and even the GPL itself (GNU General Public License 2007). Additionally, virtually every introductory how-to book for Linux I've ever seen begins with a chapter devoted to introducing the reader to the open source and/or free software "philosophy." Also, several outside academics have written normative essays & books on the revolutionary potential of hacker culture, such as Peka Himanen's *The Hacker Ethic* (2001).

Histories: Several academics and journalists have published historical accounts of computing: Glynn Moody's *Rebel Code* (2001), Steven Weber's *The Success of Open Source* (2004), Steven Levy's *Hackers* (2001/1984).

Biographies: Key figures such as Richard Stallman (Williams 2002) and Linus Torvalds (Torvalds 2001) have either written autobiographies or authorized biographies.

3.2.2 High-Impact Hubs

While the web enables great decentralization of publication, most domains tend to have a small handful of high-traffic sites and then a long-tail of smaller outlets. (For example, in the realm of political blogs, sites such as Daily Kos or Red State, on the left and right respectively, receive a lot of traffic.) The world of computing is, of course, no exception to this. "Social news" aggregation sites, such as Slashdot and the relative newcomer Hacker News, are good daily barometers of what issues are attracting a lot of attention among those in these communities. The basic structure of these sites—and more general purpose sites such as Digg—is to allow members to post links to other sites and then discuss those links. As a consequence, daily trips to Hacker News, for instance, resulted in a great diversity of primary sources, illustrating my loose “follow the crowd” sampling strategy I describe above as opposed to setting out with a set list of sources.

In addition to aggregator sites, there are a group of popular, influential opinion shapers in the various sub-communities I examine. For instance, within the Mac community, John Gruber’s blog, *Daring Fireball*, is a must-read and frequently evokes discussion around the entire community. Likewise, in the open source community, figures such as Linus Torvalds, Richard Stallman, or Mark Shuttleworth are vocal figures whose talks and writing attracts widespread attention and discussion.

Additionally, several podcasts have become high-profile outlets for deliberation in recent years. Technology podcasting networks (such as TWiT.tv and 5by5.tv), that release podcasts about technology at a daily, or greater, rate, have thrived. From open source (TWiT’s *FLOSS Weekly*), Apple (5by5’s *Talk Show*, with John Gruber), or Google and Android (TWiT’s *This Week in Google* and *This Week in Android*, respectively), podcasts have become a popular outlet for discourse about computing in recent years.

3.2.3 Community Websites

In contrast to the “elite” discourse found on popular blogs and podcasts, community websites were also an important component of my data collection. The primary means of community communication within most open source projects are online discussion boards and/or public mailing lists (I’ll refer to these collectively as “forums” here just for ease of use).

For example, Linux distribution forums and then forums for specific projects at the center of my case studies. First, Linux distributions are a natural choice because, by their very nature,[1] they are a hub for the FLOSS community and are often the main entry point for newcomers to Linux. Most Linux distributions have distro-specific forums for technical support as well as for community-building

3.2.4 News

News coverage is another key source. In particular, news coverage is important for offering the following perspectives:

Financial analysis: Much of technology journalism is really a subset of the business press. Mergers, stock prices, the financial success or failure of product launches, financial forecasts by analysts—these issues are front-and-center in most media coverage of the technology industry.

Public statements: Just as much of political journalism has sadly become more about analysis of political campaigns and strategizing than about issues & policy, much of technology journalism is analysis of the investor & public relations strategies of corporations. Press releases and public statements are mined for hidden meaning and significance.

For example, when tech journalism focuses on open source projects such as Linux or Android, they engage in the moral discourse around "open source" and it's place in the market and community. Sources for this coverage include the technology sections of major media outlets (such as The *New York Times*, *Wall Street Journal* and *Forbes*—again, the financial outlets tend to provide the most extensive coverage) as well as the smaller, technology and niche-specific news outlets such as *Linux World*, etc.

3.3 Thinking About My Data

As my research evolved in the manner described above, my approach developed into a hybrid of two approaches. On the one hand, I've collected "data" in the form of online discourse. I had no access to privileged "insider" data: everything I draw on in my analysis is publicly available. Armed with Google and a basic knowledge of how open source projects work, one can track

[1]Linux distributions pull together the products of hundreds of independent FLOSS projects and package them together into a single, coherent product: the Linux distribution.

down every resource I cite and find additional data to confirm or challenge my citations here. On the other hand, my standpoint towards my "data collection" drifted more towards that of a participant/observer or ethnographer. I ended up following Goffman's advice (1989) to "forget about being a sociologist" and experience the unfolding events as a user, contributor, and fan, of various open source projects—one who is trained as a sociologist and tends to collect text. As Wacquant (2009) puts it:

> ... "go native" but "go native armed," that is, equipped with your theoretical and methodological tools, with the full store of problematics inherited from your discipline, with your capacity for reflexivity and analysis, and guided by a constant effort, once you have passed the ordeal of initiation, to objectivize this experience and construct the object.

For example, let's return to my "data gathering" script. In the way I intended to use this script—as a large dataset to feed into some sort of qualitative analysis software—it came up short. However, this isn't to say it went unused: I archived and categorized thousands of web pages, and it has been invaluable as I try to form my experiences into a coherent account of the events in open source over the past few years. However, it functions more as a personal bookmarking and note-taking collection than as a set of data that could really stand on its own. For example, I devised a markup for adding extended comments—or fieldnotes, if you will—to these files. These notes became the basis of many of my arguments. So when writing this dissertation, these files are my first stop in pulling my narrative together, but it's not the last. To return to Mark Shuttleworth's "This is not a democracy" comment cited above: that page is in the collection. But this comment made waves, and I did not attempt to incorporate every response into the database—for one thing, the response is ongoing—nor have I limited myself to resources added to the collection as I write up that section of this dissertation. This would be a silly limitation: trying to shoehorn my analysis into a model of data collection that isn't very well-suited for my project. As an "armed sociologist," I cannot help but follow the trail through specific incidents such as this with an eye towards contextualizing the meaning of "democracy" in this context with a larger cultural discourse about democracy, participation, and decision making.

But this begs the question: does this approach limit the validity of my data? This is a common challenge for ethnography and participant-observation. With statistical analysis, or discourse analysis, other researchers can, in principle, attain the same dataset and redo the analysis to verify, or challenge, the results. Redoing an ethnography, however, is not so simple: ethnographies take place at specific times and places, and are often built upon the unique experiences and fieldnotes of a single researcher. Even if fieldnotes are made available, these are already filtered through the viewpoint of the researcher. There are three responses to this.

First, the aim of this project, as with much qualitative work, is to "craft a composite image" (Ragin 1994) of the moral culture of computing. As I've argued, this composite image will be a starting point for future research, aiming to offer an orienting frame for future social research projects. The central argument of this dissertation will be to assert that the moral culture of computing offers, for sociologists, a case of two overlapping moral conflicts over group boundaries and moral responsibility, and over the relationship between community goods and the market. I will, of course, offer evidence and make arguments about how these moral conflicts are played out in practice, but my fundamental goal is to introduce an orienting frame for thinking about the field of computing in a sociological way. The validity of this frame will, in part, be tested over time in it's usefulness in guiding future research. This dissertation only begins that project.

Second, as with any portrayal of a community and culture, the portrayal can be challenged by others who immerse themselves in that community, whether or not they have access to the exact same experiences as me. My analysis may not be "replicable" in a clear sense, but it's definitely open to challenge. If my claims are a byproduct of highlighting cases that do not represent the larger culture, then cases that contradict my framework should be widely available. For instance, I argue that deliberation over how to protect the moral purity of code from the profane market is a common theme in discussions among open source participants. If this is not the case, then prominent examples of open source projects that do not spend time deliberating over these issues on mailing lists, forums, and blogs should be easy to find. I argue, of course, that my cases are limited only by my time and resources, and, in fact, I could have chosen from a large number of cases—many of which I point to throughout the dissertation—and come to the same conclusions. This can be challenged empirically.

Finally, again, the events I've selected in this paper did play themselves out in public. If my interpretation of these events "in the moment," as I chose to write this, is wrong, then a future study looking back at these same events can challenge my interpretation. I fell short of my initial goal of building a comprehensive dataset of open source discourse, but the source material is still out there. Therefore, one could, in a sense, "replicate" my research to challenge or verify my findings.

3.4 Who is "The Community"?

In this project, I focus on free and open source software and the communities that produce this software, in particular. In other words, I am not studying the use of technology by non-developers, nor am I studying the development of proprietary software by companies like Microsoft. When I began this project, I felt very clearly about these distinctions. However, a couple of complications arose as I began my research.

First, as I'll show in Part II, "free software" wasn't coined until the mid-1980s and "open source" until the late 1990s, but the moral framework for thinking about computing was in place by the early 1980s, and open source and free software are best understood in dialogue with a larger moral field that has developed since the 1950s.

Second, not only are the lines between free software and open source software blurred, but the lines between proprietary software and both are quite fuzzy. Even companies like Apple that produce closed source *products* frequently make heavy use of, and contribute to, open source *tools*. These closed source products are also frequently dragged into the discourse of "open" technology, and the complicated, ambiguous meaning of this phrase. For instance, my chapter on usability later in the dissertation will explore the question of "openness" and Apple's iOS.

So, in the broadest sense, my subject is all of computing culture, but my primary focus is on what a subpopulation—and the discourse about freedom and openness they have fostered—tells us about this larger phenomenon.

Within the boundaries of this admittedly fuzzy community, there are implications of my methodological choices that must be addressed, as well as demographic features of the population that must be considered.

As with any community, there is an important distinction between elite and non-elites. The statements of project leaders and influential figures have a large impact in the world of open source software. While open source software projects are relatively transparent and open to participation, it is nonetheless true that most open source projects have either a single leader or a handful of leaders who make the key decisions about the direction of the project.

Additionally, a small number of figures (such as Mark Shuttleworth, mentioned above) have become "superstars" of the open source world, making waves and shaping opinions.

Additionally, online communities require a distinction between *participants* and non-participants, or *lurkers*. Also, perhaps confusingly, participants for my purposes are those who contribute to online discourse, not who contribute code. Many contribute code without participating much in online discussions. The reverse is also true: many vocal contributors to online discourse around open source software are not programmers, but journalists, bloggers, or others who are perhaps primarily interested in the moral and political, rather than technical, issues.

A bias towards elites and vocal participants is a limitation of my aforementioned decision to focus on existing discourse rather than conduct, say, original interviews. Programmers who contribute code, but avoid contentious debate over moral and political issues, in particular, are an interesting sub-population for future research. However, as I've argued throughout this chapter, future research of this "silent" members of the community will be better framed against a backdrop of the more public, vocal culture of computing that I'm focusing on here.

There are also clear demographic features of this population to consider. For instance, gender differences in computing, and open source computing, in particular, has received a great deal of attention. A 2002 survey of software developers (Ghosh et al. 2002, Ghosh 2005) had the rather shocking finding that while women comprise 28% of proprietary developers, only 1.5% of open source and free software developers were women. A follow-up interview project (FLOSSpols 2006) argues that the culture of open source communities, in particular the informality of the culture, reinforces gender differences and enables the perpetuation of stereotypes:

> F/LOSS communities actively perpetuate a 'hacker' ethic, which situates itself outside the 'mainstream' sociality, but equates women with that mainstream. Women are treated as either alien Other or (in online contexts) are assumed to be male and thus made invisible. Women are seen as innately more able to organise, communicate and negotiate among F/LOSS projects as well as with the outside world. Thereby they become carriers of sociality that is seen in a contrast to the 'technical' realm ascribed to men. Additionally F/LOSS women receive a high level of attention due to their gender which decreases their feeling of acceptance as community members as well as their willingness to further engage with the

community.

In contrast to the informal, voluntary nature of open source projects, participation in proprietary software development generally involves doing so professionally, for organizations that have institutionalized hiring practices that highlight skill and de-emphasize informal cultural capital irrelevant to actual software development.

Participation in computing culture obviously requires access to computers, and, as computing culture is tightly connected to internet culture, it also requires internet access. The FLOSSpols survey, for instance, finds that young women in their survey were less likely to have their own computers growing up compared to their male counterparts, and that having a computer of your own at a young age is a strong predictor of participation in open source projects. In addition to gender differences, access to computers and internet connectivity has become more pervasive over time, but inequality of access to technology also entails a bias towards upper and middle class segments of the population. Additionally, there's a racial and national dimension to this unequal access to computing, with white American and Europeans being over-represented in the first wave of computing culture. However, the rapid growth of technological globalization makes this a fast-changing area, and it is an interesting empirical question as to whether open source projects today are more diverse than they were a decade or two ago.

In the next section, I turn towards showing the emergence of the moral field during the history of computing the second half of the twentieth century. Following that, I examine contemporary case studies of the moral field in action. In each of these pursuits, I aim to build a consistent, coherent narrative account of the ways in which culture, social boundaries, and interests interact to make software development a pursuit laden with moral and political deliberation. As I argued above, I believe such an understanding is needed and will provide a foundation for many interesting empirical and theoretical developments on the subject.

Part II

History of the Moral Field

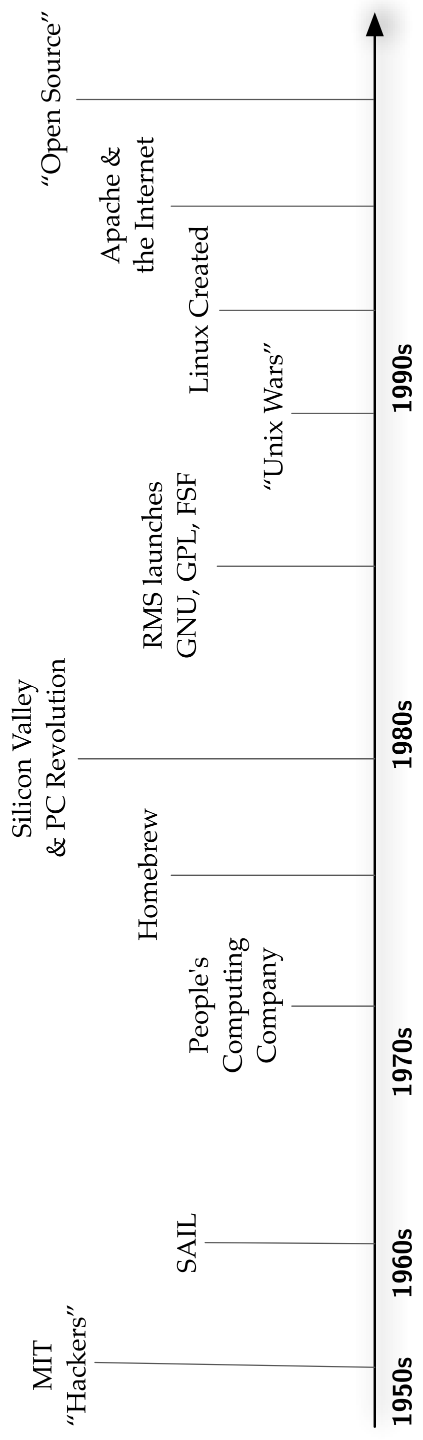
Chapter 4

The Moral Field Emerges

Chapter Preview: The moral field of computing emerged in the second half of the twentieth century. Its shape today is a product of the social, cultural, and economic circumstances under which computers and computer programming developed. This chapter traces the development of the moral field from the small communities of early hackers to the open source boom of the 1990s and 2000s. When viewed from the perspective of the moral field, this history is understood as a history of negotiation between competing visions of markets and community, and between competing visions of computing as an expert culture and computing as a public good.

Many books have been written about the social and cultural history of computing (to name just a few: Levy 2001/1984, Markoff 2006, Moody 2001, Salus 1994, Weber 2004, Berry 2008). By and large, a handful of core events are identified in most histories of modern computing, which I've represented in the timeline in Figure 4.1. My aim in this chapter is not to offer a comprehensive overview of this history, but rather to briefly recount some of these core historical events and show how viewing them from the perspective of cultural sociology reveals patterns and themes that help illuminate the shape of the moral field of computing. I will show how the moral field I introduced in the previous section emerged over time, and how it has shaped interaction in computing in recent history.

Figure 4.1: Historical Timeline



4.1 Hackers and Expert Communities

The earliest computers were not very accessible. They were large, expensive, and rare. Programming involved punching a series of holes in cards, handing those cards off to the only people allowed to be in the room with the computer—certified administrators who came to be known as "The Priesthood"—and waiting hours or even days to get the results. Most people had a fairly narrow vision for computers as simply number-crunching machines. The math department at MIT referred to computer science as "witchcraft" and warned its students that wasting time on computers would only turn one into a clerk (Levy 2001/1984:76).

4.1.1 MIT Hackers

At MIT, however, a group of students who called themselves "hackers," along with some supportive faculty such as John McCarthy and Marvin Minsky, co-founders of Artificial Intelligence (AI), had a much bolder vision.[1] Working around the clock at MIT's AI lab, early hackers made their PDP-1's function as calculators and typewriters, play music and run programs for playing chess and games with flying spaceships. Instead of handing off punchcards to The Priesthood, hackers articulated a "Hands-On Imperative," the belief that one learns about the world by taking things apart and putting them back together (Levy 2001/1984:40-46). They valued hands-on interaction with the machines and challenged one another to find ever more innovative and elegant ways of pushing the limits of what the computers could do. They elevated computers from mere industrial tools to objects at the center of a rich, complex subculture.

A "hacker ethic" evolved out of the Hands-On Imperative (Levy 2001/1984, Himanen 2001). Information, necessary for diagnosing problems and making improvements in any type of system, ought to be free. Authority, in particular large bureaucratic authority, was viewed as an obstacle to this free exchange of information. The central nemesis in this war: IBM and its bureaucratized model of computing-by-Priesthood. Early hackers theorized that the giant, inflexible IBM machines were a reflection of the social structure that produced them: rigidly bureaucratic and button-down. Hacking, on the other hand, was about creativity and flexibility. The beloved PDP series of computers in use at the AI Lab embodied this

[1]In popular culture, "hacker" has taken on more negative connotations in recent decades. In this paper, hacker refers to the original, positive meaning. For those interested in how these two, contradictory images of hackers came into being, Nissenbaum 2004 is a good starting point.

openness and flexibility. Hacking was about meritocracy—your programming skill was the source of all prestige in the community—and programming was not the mindless management of technology, but a source of creativity, art, beauty and, for some, social change and the faith that:

> If *everyone* could interact with computers with the same innocent, productive, creative impulse that hackers did, the Hacker Ethic might spread throughout society like a benevolent ripple, and computers would indeed change the world for the better. (Levy 2001/1984:49)

As computer technology evolved, the hacker ethic responded. One of the obvious shortcomings of computers like the PDP-1 was that it was a single user machine: one person ran the show at a time while others watched over the shoulder or hand-wrote their own program while awaiting their time in the driver's seat. In the 1960's, one of the great breakthroughs in computing was the introduction of *time-sharing*, or multi-user, machines. By attaching several terminals to a single computer, multiple users could sit at their own terminals and run their own programs simultaneously, with the computer managing all the processes behind-the-scenes.

You might think hackers would welcome this development—more people could use the computer at once, right?—but in fact hackers found the implementation of time-sharing at the time, the one in place at MIT dubbed the Compatible Time-Sharing System (CTSS), to be antithetical to the hacker spirit. CTSS required each user to have their own account and password, and was also much slower because of sharing the computer's resources with multiple users at a time. Perhaps worst of all, the administrators limited and charged for access based on time, memory usage and disk space. Where the hacker ethic encouraged openness, CTSS encouraged privacy; where the hacker ethic encouraged Hands-On tinkering that pushed a machine to its limits, time-sharing sequestered users away from the computer at their own terminal and charged them if they spent too much time tinkering (Levy 2001/1984:120).

The benefits of time-sharing were too large though, but rather than adopt an existing time-sharing operating system like CTSS or Multics, hackers at the AI Lab built their own time-sharing system, which they playfully called ITS, or "Incompatible Time-Sharing System," to poke fun at CTSS. Rather than leveraging the system's multi-user abilities to increase privacy and isolation, ITS built upon a culture of sharing. From the vantage point of today's computer security, the deliberate lack of security in ITS is shocking. ITS users had no passwords and all users could access one another's files. Just as hackers used to look over one another's shoulders prior to time-sharing, ITS included a system of screen sharing where a user at one terminal could have another user's terminal displayed on their own screen. Just as hackers used to have cabinets full of paper tape that anyone could access to learn from the work others had done, ITS users could access the personal files of all other users. ITS was an astonishing collective achievement, assembled piece-by-piece by hackers who sought to instill the hacker ethic into the system. Hacker Don Eastlake described ITS' development like this:

> In general, the ITS system can be said to have been designer implemented and user designed. . . Features are less likely to turn out to be of low utility if users are their designers and they are less likely to be difficult to use if their designers are their users. (Levy 2001/1984:127)

The halcyon days at MIT's AI Lab did not last, however. More institutions acquired computers and other hacker hotbeds sprung up around the country, notably at Stanford, who managed to pull John McCarthy out West to start the Stanford AI Lab (SAIL). Business was increasingly in the mix as well. Xerox'x Palo Alto Research Center (PARC), also on Stanford property leased from the University, became a center of computer innovations: the Graphical User Interface (GUI), bitmap graphics and object-oriented programming, to name just a few, all began at PARC. Just as their peers in the 60's counterculture were transitioning into white-collar jobs and settling down in the suburbs, die-hard first-generation hackers gradually left the AI Lab for lucrative jobs in the brand new IT industry.

As AI Lab alumni, and their fame, spread throughout the country though, the hacker ethic spread as well. The sharing ethos of ITS expanded even further with the introduction of ARPAnet, the predecessor to today's internet, which enabled hackers from all over the country to connect to the ITS system built at MIT and create a national community of hackers. Additionally, a vision spread of computer programmers not as dull engineers but as craftsmen and artisans.

4.1.2 Unix

During the 1960s and 70s, the hacker culture spread to other universities, Stanford (particularly SAIL) and Berkeley on the West Coast became Hacker hotbeds, and also into corporate research labs. For example, the invention and growth of the Unix operating system began at

Bell Labs in 1969 when researchers Ken Thompson and Dennis Ritchie decided to revive the abandoned Multics project "just for fun," and began work on the Unix operating system (Salus 1994).

Forty years later, Unix and its descendants are now running on everything from mainframes to laptops to cell phones. Descendants of Unix—from Linux to Mac OS X—will be central topics throughout this dissertation. The technical and cultural legacy of Unix will be instrumental to the rest of this story.

Thompson and Ritchie built Unix with a specific design philosophy in mind (Salus 1994, Raymond 2003). This philosophy was most famously expressed by Douglas McIlroy, a programmer responsible for developing several foundational Unix tools:

> "This is the Unix philosophy. Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface."

Perhaps the best way to illustrate this philosophy is by taking an example of software most academics today are intimately familiar with that is, in many ways, the antithesis of the Unix philosophy: Microsoft Office.

Microsoft Office is a single package of software consisting of a word processor (Word), a spreadsheet application (Excel) presentation software (PowerPoint), and other supporting programs. Each of these applications store their data in their own unique file format: .doc files (Word), .ppt (Powerpoint) and .xls (Excel). Over the years, Microsoft has added many features to Office: a spell checker, a thesaurus, a grammar checker, clip art, multimedia support, a commenting system and "track changes" feature for collaborators, and, of course, an animated paper clip aimed at helping you find your way around these various features. Each of these tools exists only within the Office universe. Everything you might want to do with these files—from editing to printing to spell-checking to merging and converting—all takes place inside Office. To share these files with others, they need the complete Office suite as well (and preferably the same version as you). Office is a monolith. As technology evolves and new features are required of Office (from multimedia support to internet connectivity), these features get tacked on to Office, making it ever-larger and slower, and making its users more and more locked in to the Office ecosystem.

To translate Office into the Unix philosophy would require some heavy redesign.[2] Each of the components—not just Word, Excel and Powerpoint, but the spell checker, searching capabilities, multimedia library, versioning system, etc.—would have to be broken up into their own independent utilities. Instead of being a part of one huge monolith, the utilities would have to communicate with one another.[3] The third principle of Unix design is the most radical, and most distinctive to Unix: every application communicates via plain text. Unix programs output plain text files which can be accepted as input into any other program. The result of this is that Unix users can chain Unix utilities together, *piping* the output of one program into another. The classic Unix interface, the plain text command line, remains in wide use today, despite its anachronistic appearance to the casual computer user, precisely because of its inherent flexibility and scalability: the ability to take the basic Unix building blocks and construct highly complex workflows and systems.

The design emphasis on small, clean utilities and modularity was not just an aesthetic choice. It facilitated a collaborative, decentralized division of labor from the start. If your program took text streams as its input and resulted in compatible text streams as output, then it could be dropped in seamlessly with the rest of the Unix system. Like its predecessor ITS in the MIT AI Lab, the structure of Unix invited individual tinkering and collaborative, but decentralized, systems design. Again, the social structure developing and maintaining the technological system was inherently linked with the technical design of the system (Weber 2004).

Unix's spread was also in large part due to an accident of birth: it's founding in Bell Labs (BTL) at the precise time the federal government was taking action against AT&T under the Sherman Antitrust Act. AT&T was forbidden from profiting from sales outside of the telephone, telegraph or "common carrier communications" industry. "Common carrier communications" is a vague phrase open to interpretation, and its shifting interpretation by federal regulators and AT&T lawyers explains much about the history of Unix (Salus 1994, Weber 2004:22). Early on, the phrase was conservatively interpreted and, fearful of commercializing Unix, AT&T decided to license Unix, complete with source code, for a

[2]Regretably, this hasn't been the path taken by most Unix office suites, which have tended to simply clone the functionality of Microsoft Office.

[3]Mac OS X's system services are an attempt to replicate this aspect of Unix in a modern graphical environment: for example, a users photo and music library is accessible to other applications, and all OS X applications use the same system-wide Dictionary and spell-checker.

nominal fee. This led to widespread Unix adoption and a growing community of programmers contributing to the platform. AT&T provided no support, so community-based support and development flourished in their absence. Universities adopted Unix and an entire generation of programmers learned programming on Unix and the portability of Unix's source code—written in the C programming language—allowed users to port Unix to a wide range of machines. In short, Unix was, in effect, "open source" before the "open source software" existed. The technical design of Unix and the social conditions in which it was distributed created a community of user/programmers devoted to the platform. Salus (1994:143) describes this process:

> Something was created at BTL. It was distributed in source form. A user in the UK created something from it. Another user in California improved on both the original and the UK version. It was distributed to the community at cost. The improved version was incorporated into the next BTL release.
>
> There was no way that Patent and Licensing could control this. And the system got better and more widely used all the time.

4.1.3 Homebrew

In the 1970's, technological advancements began to bring computing out of large corporations and universities. Declining hardware prices led to a growing community of "hobbyists," who spent their evenings and weekends scavenging parts stores and soldering together all varieties of gadgetry. In short, *hardware hacking* was now accessible to anyone with the time and patience.

When a tiny company in Albuquerque, New Mexico decided to take the new microprocessors Intel had developed and assemble a home computer cheap enough for anyone to buy. The result was the Altair 8800. The Altair, to an outside observer, did not appear to be all that great. It had only 256 bytes of memory and was simply a box with a set of switches and lights on the front. No display, no keyboard, no storage devices. But it was a computer, and it was a smashing success. It was expandable—an industry quickly sprung up selling additional parts for the machine—and it was eminently hackable. Its arrival on the scene helped spark the formation of a group whose only rival is MIT's AI Lab in its impact on hacker culture: the Homebrew Computer Club.

With the release of the Altair and the take-off of home hardware hacking, a long-present tension in the hacker community began to intensify: *hackers* versus *planners*. For pure hackers, hacking is an end in itself and the getting hands-on access to hardware was all that mattered. Planners, however, while often hackers themselves, were also wrapped up in the social applications of technology, "more absorbed by the goals of computing than addicted to the computing process," (Levy 2001/1984:66). The tension existed in the MIT lab as well, but the more explicitly political bent of the California hackers of the 70's brought the issue to the forefront of the community. The PCC and Community Memory were *political* projects, and while their founders were, of course, hackers who loved fiddling with technology, the broader social implications of the technology was a serious motivation behind their efforts.

The Homebrew Computer Club was formed by hackers who wanted to take hardware hacking further than PCC founder Bob Albrecht wanted within the confines of PCC, which Albrecht wanted to remain accessible and outreach-focused. However, Homebrew began at just the right time: right as the Altair became available, and the meeting's attendees would turn into a who's who of Silicon Valley in the next decade.

The culture of sharing and cooperation that defined the hacker ethic continued at Homebrew. Perhaps the most famous story in hacker history is Bill Gate's "Open Letter to Hobbyists," published in Homebrew newsletter in 1976. Gates, and his upstart company Microsoft, had written a version of the BASIC programming language for the Altair and was selling the software in partnership with MITS, the company behind the Altair. Following the hacker tradition of sharing paper tapes, however, a hacker brought Microsoft's BASIC to a Homebrew meeting and copies were made and circulated. The only rule: if you take a tape, bring two copies next time to give away to others. Gates responded by writing a letter:

> As a majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?...
>
> Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software...Most directly, the thing you do is theft.

Despite the ethic of sharing and aversion to the commercialization of software, Homebrew gradually transitioned from a collection of hobbyists to participants in a thriving young industry: from starting hardware companies selling add-ons for machines like the Altair to building new next-generation machines themselves, the most famous of which was the Apple computer. At Homebrew meetings, Steve Wozniak would bring prototypes of his early Apples to show off to other hackers and to solicit feedback and advice. Wozniak was the hacker and his partner, Steve Jobs, was a new breed of planner, a mediocre engineer but a visionary in terms of design and marketing. Jobs describes their relationship:

> [Wozniak] took me to some Homebrew Computer Club meetings, where computer hobbyists compared notes and stuff. I didn't find them all that exciting, but some of them were fun. Wozniak went religiously...He designed most of [the Apple]. I helped on the memory part and I helped when we decided to turn it into a product. Woz isn't great at turning things into products, but he's really a brilliant designer.[4]

Whereas other businesses cropping up from the Homebrew crowd were exclusively hacker-run, Jobs hired a professional management team to run Apple (still operating out of a garage), industrial designers to build sleek, appliance-like machines and graphic artists to design what became one of the most iconic corporate logos of all time. "Steve and Steve" personified the split personality taking over Homebrew: inward-looking geeks who loved to tinker with computers, and shrewd businessmen who saw the future of the technology industry. However, the encroachment of business into their community was met with some reservation. For example, Homebrew co-founder Fred Moore became disenchanted with the group's "seduction" by technology as well, particularly as the microcomputer industry took off (and people came to Homebrew "with dollar signs in their eyes") and Moore learned of the awful labor conditions under which hardware was being manufactured in Asia (Levy 2001/1984:216).

While we begin to see the expansion of computing culture alongside increased access to technology, Homebrew was still an expert community—firmly situated in that lower left quadrant of Figure 2.1 above—and not all that different from the hackers at MIT. However, during the 1970's and into the 1980's the full shape of the field began to emerge.

[4]playboy1

4.2 The Moral Field Expands

While early groups and projects like the MIT hackers and Unix began as expert communities, events in the 1970s and onward would push computing—and the culture of computing—in two directions simultaneously: computing increasingly became a public phenomenon, open to all of society instead of a handful of technological elite, and computing also became big business, reaching into virtually every corner of the entire economy. The moral culture that emerged in the expert communities of places like the MIT AI lab, Bell Labs, and Homebrew—emphasizing free sharing of source code and hands-on access to the lowest levels of hardware—clashed with the competing priorities of private intellectual property and a public that wanted to use, but not tinker with, computers. Approaching the events listed in Figure 4.1 from the perspective of these shifts in both markets and group access, this section shows how this one-dimensional timeline consists of events that give shape to the moral field as depicted in Figure 4.2.[5]

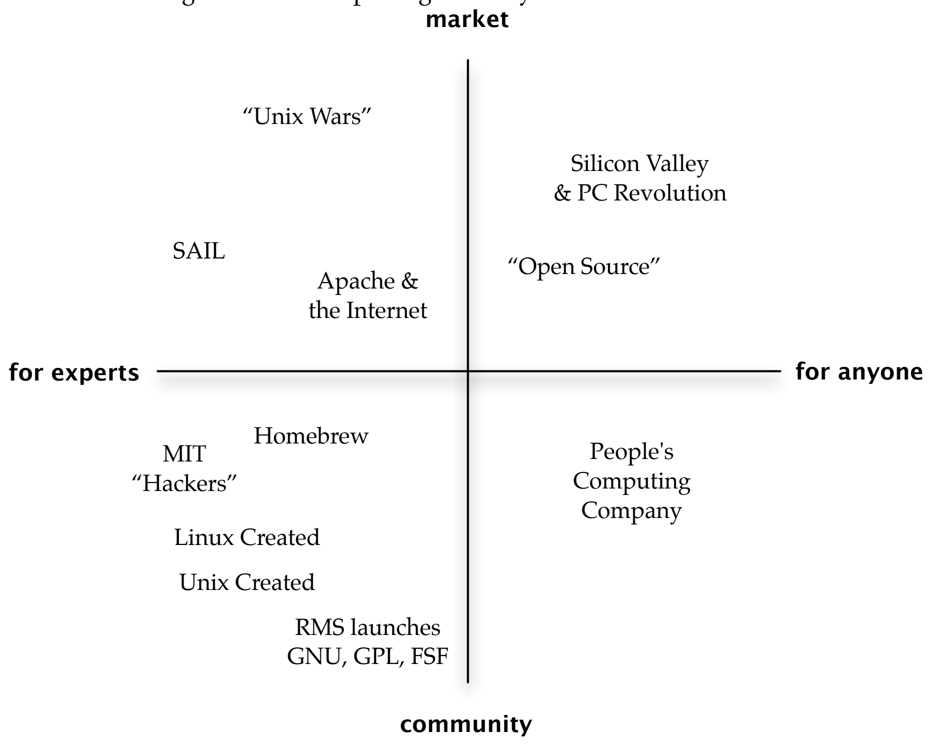
Into the Market

Computers have always been commercial products, though the role of researchers at universities and private research labs like Bell Labs enabled some distance between the commercial application of the technology. However, the market for computers expanded considerably in the 1980s and this placed pressure on several of the *communities* that had developed to begin to act like *market actors*. On the "expert" end of the moral field, nowhere is this seen more clearly than in the story of Unix.

By the late 1970s and early 80s, the antitrust environment had changed dramatically and AT&T changed course on commercializing Unix. After nearly a decade of an open, decentralized development process, this proved to be quite tricky. The University of California, Berkeley, in particular, had become a center of Unix development. In 1975, Ken Thompson took a sabbatical from Bell Labs and spent a year at Berkeley, working with several faculty members and graduate students that were extending Unix in innovative ways, creating tools such as the Pascal programming language and the vi text editor, that became staples of Unix and computer programming. By the late 70's, a graduate student at Berkeley, Bill Joy, was distributing a popular collection of Unix tools and utilities he called BSD ("Berkeley Software

[5]The positioning of events on the field in Figure 4.1 does not represent the precise mathematical output of an algorithm. It is intended as a rough illustration of the processes of cultural negotiation and positioning I will describe throughout the rest of this section.

Figure 4.2: Computing History and the Moral Field



Distribution").

When AT&T began commercializing Unix, the collaborative relationship with Berkeley soured. In place of generous, affordable licensing terms, AT&T tightened up Unix licensing: by 1984, official AT&T Unix licenses ran $100,000 or higher (Weber 2004:39). Berkeley, whose BSD had transformed from a collection of utilities into a full-blown version of the Unix operating system, spent much of the 1980s carefully stripping AT&T-owned code and replacing it with Berkeley-written code. Unlike AT&T, BSD was distributed for free under a liberal license: derivative code must give credit to Berkeley, but beyond that programmers are free to do whatever they wish with the code. AT&T and Berkeley became fierce competitors. AT&T owned the Unix trademark and only their version could be called Unix.[6] However, much of the cutting edge development was taking place at Berkeley, who won lucrative DARPA contracts to develop the TCP/IP protocol for BSD, a development which formed the backbone of the internet today.

Consequently, the 1980s saw the Unix world splinter into a variety of incompatible variants of Unix and expensive lawsuits over the Unix trademark and the ownership of Unix code. Universities abandoned Unix as the teaching tool of choice because of AT&T's skyrocketing licensing fees. The expert community created at Bell Labs had turned into an expert market, and a particularly expensive, divided, and litigious market at that. By this time, another development—the personal computer—was pushing computing culture away from the tiny cliques of experts into the broader public.

Into the Public

The utopian rhetoric and anti-authority attitude of the MIT hackers may seem, on the surface, to have had something in common with the wave of student radicalism hitting their peers in campuses across the country. In fact, the AI Lab was avowedly apolitical and isolated from the social movements surrounding the lab. Even if individual hackers were sympathetic to political movements against the war and for civil rights—and many of them were—in practice, the most significant effect of these movements on hackers was the necessity of greater security

[6]Or technically, UNIX, with all capital letters. I'm skipping this convention here simply because it is awkward to type and read, but also because Dennis Ritchie, who created Unix, prefers "Unix" as well: "The difference came because the lawyers decided that all trademarks should be spelled in upper case... Since Unix isn't an acronym, just initial-cap, which I now prefer, seemed more logical. However, one complicating factor was that when we got a phototypesetter, we were so thrilled by it that we thought it was fun to spell UNIX in small caps just to prove we could do it" (Manesh 2002).

to protect their military-funded computers from attack by protestors. The American Left of the 60's and early 70's was highly suspicious of technology, with the dystopian visions of *1984* and *Brave New World* painting technology as a tool for oppression and surveilance. The liberation-through-technology worldview of the hackers was perceived as naive and dangerous, to the extent that it was perceived at all: at this point, computers were still something only a very small minority of the population experienced hands-on. In the 1970's in California, a group of hacker-activists began to change this (Levy 2001/1984, Markoff 2006).

Decrying the "excessive purity" of the "technological Jesuits" from MIT (Levy 2001/1984:182), a new group of hackers fully immersed in the student culture at Berkeley and Stanford cultivated a more populist, politicized strain of the hacker ethic. A group of Berkeley-area hackers started a tabloid publication and walk-in center, dubbed *People's Computer Company* (PCC). The first issue's cover contained the text, "Computers are mostly used against people instead of for people. Used to control people instead of to FREE them. Time to change all that—we need a. . . People's Computer Company." An influential book at the time was called *Computer Lib*, whose hand-drawn cover boldly pronounced, "You can and must understand computers NOW" and promoted widespread "computer literacy." *Computer Lib* author Ted Nelson called the Homebrew attendee's "chip-monks," and Albrecht said, "I could only understand about every fourth word these guys were saying" (Levy 2001/1984:220). Homebrew co-founder Fred Moore became disenchanted with the group's "seduction" by technology as well, particularly as the microcomputer industry took off (and people came to Homebrew "with dollar signs in their eyes") and Moore learned of the awful labor conditions under which hardware was being manufactured in Asia (Levy 2001/1984:216).

The hackers behind the PCC also enacted an ambitious project called "Community Memory": a series of networked terminals around the city that made up the first electronic bulletin board, used for everything from classifieds to sharing poetry and jokes. The PCC and Community Memory were *political* projects, and while their founders were, of course, hackers who loved fiddling with technology, the broader social implications of the technology was a serious motivation behind their efforts:

> [Hackers considered] the computer itself a model for activism, and hope the proliferation of computers to people would, in effect, spread the Hacker Ethic throughout society, giving the people power not only over machines but over political oppressors (Levy 2001/1984:181).

Into the Mass Market

To the chagrin of early hackers, Gates' view largely won out in the 1980s and 90s and the dominant companies, such as Apple and Gates' Microsoft, embraced a *proprietary* model of software development and distribution.

Take Microsoft's Windows operating system, for example. The platform is owned and controlled by Microsoft. The source code for the Windows operating system is maintained internally by Microsoft engineers and is not available for others to view and modify. If you want to develop software for Windows, you'll use the Application Programming Interfaces (APIs) developed by Microsoft. If you find a bug or a limitation in these APIs, you can submit a report to Microsoft and hope they fix it. When Microsoft releases updates to Windows, you pay Microsoft to get access to the updates. When Microsoft adds a feature to Windows—such as networking or graphics capability—or creates file formats such as Microsoft Word's ".doc" format, they do so in a manner that ensures compatibility with Microsoft software, but nothing else.

In short, the Windows platform carries particular kinds of social relationships and cultural expectations that are anathema to the hackers from the MIT AI Lab or the Homebrew Computer Club. The emergence of platforms such as Windows transformed software from something that is to be shared within a technocratic community of expert hackers into a product designed is to be bought and sold to consumers in the market.

When viewing the events in Figure 4.1 from the perspective of the moral field (Figure 4.2), key events in computing history can each be understood in terms of their impact and position within this field. The MIT Hackers' rebellion against IBM and corporate bureaucracy in favor of a tight-knit group of elite hackers represent a shift towards the lower-left quadrant of the field. The "People's Computing Company" pulled activity away from a tight-knit group of "chip-monks" into the lower-right quadrant, taking the community model of computing culture to the masses. The "Unix Wars" of the late 1980s taking Unix out of it's community roots and into a environment of competing, proprietary, big business-oriented vendors and Unix-based OSes shifted activity upwards into the upper-left quadrant. The PC revolution and Microsoft's mission of a "PC on every desktop and in every home" took the pro-market, proprietary model of computing to the masses in the upper-right quadrant.

Chapter 5

Free Software, Open Source Software

Chapter Preview: In the mid-1980s and 90s, the moral field was fully developed and the free software and the open source software movements emerged out of negotiation with this field. The tensions between markets and community led Richard Stallman to found the free software movement, but the commercial utility of Linux and other "free software" led to the coining of an "open source" movement. After this chapter, the stage is set for the contemporary case studies discussed in the following chapters.

By the mid-1980s the basic structure of the moral field was established. There were communities, corporations, professionals, hobbyists, and consumers. There were competing visions of computers as a democratizing tool for human liberation, and computers as the next big economic wave to ride straight to the bank. And hackers existed across the moral field. At Microsoft, hackers saw proprietary source code not as a violation of a sacred ethic, but from the perspective of "The Liberal Dream": as an efficient, elegant way of achieving their own moral vision, that of "a PC on every desk," empowering all of society with access to computers.

5.1 Richard Stallman and the Free Software Movement

Not all hackers were persuaded, however. In the early 1980s, MIT AI Lab hacker Richard Stallman had grown frustrated with the increasing commercialization of computing and the restrictive, secretive policies companies were adopting for their code. Stallman believed these developments were not just inconvenient, but were immoral and unjust.

Stallman was widely known for creating the popular Emacs text editor. Emacs began as a collection of macros that allowed a user to customize and add features to the TECO text editor. A community of hackers began using Stallman's macros, improving them and creating their own. Over time, Stallman took the most popular macros and unified them into a single editor, Emacs. Stallman decided to distribute the editor for free, with one condition: contribute any changes made back to Stallman so he could improve the editor, an arrangement Stallman called the "Emacs Commune" (Williams 2002). Taking the lessons he'd learned from the Emacs Commune and his mentors in the AI Lab, Stallman quit his job at MIT in 1984 and went to work full time on building a completely free implementation of Unix. Following hacker tradition of playful naming, Stallman called his operation system GNU, a recursive acronym for "GNU's Not Unix," poking fun at the fact that Unix was trademarked by AT&T so he couldn't officially call it a "Unix."

Stallman saw the development of proprietary software as antithetical to the hacker ethic, but also immoral, violating not only the scientific ethic of academic computing but also the golden rule (Williams 2002). The moral nature of Stallman's dissent is clear in his language. The non-disclosure agreements programmers are forced to sign to keep source code secret are "anti-social" and "unethical," making "the first step in using a computer [a] promise not to help your neighbor" (Stallman 1999). Stallman likes to compare software to cooking, thinking of source code as recipes:

> So imagine what it would be like if recipes were packaged inside black boxes. You couldn't see what ingredients they're using, let alone change them, and imagine if you made a copy for a friend, they would call you a pirate and try to put you in prison for years. That world would create tremendous outrage from all the people who are used to sharing recipes. But that is exactly what the world of proprietary software is like. A world in which common decency towards other people is prohibited or prevented (Stallman 2001).

Stallman vowed to fight these "fascist advances with every method I could" (Levy 2001/1984:419), and is responsible for a series of steps that laid the foundation for today's free/open source ecology: launching the GNU Project, founding the Free Software Foundation, and creating the GNU General Public License. Together these projects promote what Stallman called "Free Software." As we've seen, the notion that code should be freely shared between hackers and that charging for software while keeping source code secret had a long tradition in programming culture, Stallman's contribution was to give this tradition a name, institutionalize the practices of free software and to develop a formal, written philosophy of free software.

The "free" in free software stands for "liberty" or "freedom," not for "without charge." *Libre*, not *gratis*. "Free as in speech, not beer," is a popular expression of this distinction. Stallman, and those he recruited to his cause, began to build, piece by piece, a free operating system. And perhaps the most influential thing Stallman would write, however, was not software but a legal document: a software license he dubbed the GNU General Public Licence (GPL). The GPL takes Stallman's four freedoms and wraps them inside a legal license. Whereas copyright is traditionally viewed as a protection for the content producer, the GPL uses copyright for the purpose of protecting the content consumer, a twist Stallman calls the "copyleft." The GPL, to put it as simply as possible, states that a user of GPL'd software has each of the four freedoms outlined above on one condition: that if they redistribute and/or modify the code, they must do so under the GPL or a "compatible license" that grants future users those same freedoms. In other words, like traditional licenses, the GPL places restrictions on what end users can do (they cannot redistribute under a more restrictive license) and grants users privileges (the four freedoms) but the nature and scope of these privileges are radically different than a traditional copyright. Stallman has described the copyleft as "a form of intellectual jujitsu, using the legal system that software hoarders have set up against them" (Williams 2002). Like the name GNU, the GPL is a "clever hack" of the legal system.

In Stallman, we see a melding of the two hacking traditions: the elite hackers of MIT and the populist, politicized hackers of the PCC era. Like both traditions of hackers, Stallman represents a tension between the "hackers" and the "planners." No one doubts Stallman's skill as a hacker, but he emphasizes the social and political aspects of hacking culture in a way his predecessors at MIT resisted. For instance, Stallman makes a point of explaining that he will choose an inferior free program over a technically-superior proprietary application:

> "I think that freedom is more important than mere technical advance...I would always choose a less advanced free program rather than a more advanced non-free program, because I won't give up my freedom for something like that. My rule is, if I can't share it with you, I won't take it" (Stallman, in Williams 2002:Ch 8).

Stallman has a reputation as a hard-liner with respect to what counts as "free software." The GPL is the most famous open source license, but in fact, there are several "open source" licenses with different philosophies. The open source license most often contrasted with the GPL is the BSD License, which the University of California adopted for it's BSD distribution of Unix. The BSD license is, in one sense, "more free" than the GPL in that it places very few restrictions on users: do whatever you want with the software, just make sure to include credit to the University of California. So companies can take BSD-licensed code, make modifications and release the result under any license they want. Apple's proprietary Unix operating system, Mac OS X, for example, is built on top of BSD Unix.

5.2 Linux

In the early 1990's, the Unix world was a relatively bleak place compared to the rest of the computing industry. PC's were starting to take off, and Microsoft's Windows operating system was dominating both the growing business and consumer markets. While Microsoft was cleaning up, competing Unix vendors were busy taking legal action against one another and building proprietary elements into their own flavors of Unix, creating a mess of incompatible Unix systems. Berkeley's BSD Unix, despite years of rewriting from scratch to remove any proprietary code, was tangled up in lawsuits as well. Stallman's GNU Project had successfully produced many components of the GNU operating system, but had yet to complete the GNU kernel, which as the name suggests, is at the core of the operating system, interfacing between the computer's hardware and the other components of the system.

In the 1990s, the free operating system would find it's kernel at last, but it would not come from Stallman and the GNU project, but from Finnish student, Linus Torvalds. Torvalds simply wanted to be able to follow his email and newsgroups from the comfort of his dorm room during the cold Helsinki winters. So he decided to build his own version of Unix to run on his personal computer. Like Ken Thompson two decades earlier, Torvalds began this project "just for fun," as he would title his autobiography (Torvalds 2001). However, whereas Unix spread slowly over the course of a decade via the passing of tapes between a handful of Universities with computers, Linus's Unix (which would become "Linux" for short) came into being right as the internet was coming into its own.

In 1991, Torvalds announced his new project on the mailing list of an education-based Unix variant, Minix, and shortly thereafter posted version 0.01 on a public ftp server and it quickly gathered steam. The initial versions of Linux were licensed under terms Torvalds crafted himself, but he switched Linux to the GPL relatively quickly. Fearful of commercializing Linux, but wanting to make accomodate certain kinds of commercial transactions, the GPL was an attractive option. Additionally, Linus admits some loyalty to the GNU project, early on:

> The fact is, to make Linux usable, I had relied on a lot of tools that had been distributed freely over the Internet—I had hoisted myself up on the shoulders of giants. The most important of these free software projects was the GCC compiler (GNU C Compiler). It had been copyrighted under the General Public License. . . Under the terms of the GPL, money is not the issue. You can charge a million bucks if somebody's willing to pay it, but you have to make the sources available. And the person you give or sell the source to has to have all the rights you have. It's a brilliant device (Torvalds 2001:95-6).

In 1991, Torvalds had attended a talk by Stallman at the University of Helsinki. From the beginning the practical benefits of the GPL were obvious to him, but not the political bent of Stallman and the GNU project:

> Judging from the fact that I don't remember much about the talk back in 1991, it probably didn't make a huge impact on my life at that point. I was interested in the technology, not the politics (Torvalds 2001:58).
>
> [. . .]
>
> Richard Stallman wants to make everything open source. To him, it's a political struggle, and he wants to use the GPL as a way to drive open source. He sees no other alternative. The truth is, I didn't open source Linux for such lofty reasons. I wanted feedback. And it's how things were done in the early days of computers, when most of the work was done at universities or defense establishments and they ended up being very open (Torvalds 2001:194).

In just a few short years, Linux would go from being a fun side-project starting in a Helsinki dorm room to a full-fledged operating system competing with the Microsoft's and IBM's of the world. The social structure and norms that guided this quick development echo the history of hacker culture.

In the language of the MIT hackers, Torvalds was emphatically not a planner. Linux was not a platform to test groundbreaking ideas in computer science. Rather the overarching goal was digging in and making things work: "Make it work first, then make it better" (Moody 2001:80). Two quick examples of key early design decisions illustrate this example.

First, without going into technical detail, there are basically two routes towards developing an operating system kernel, a "monolithic kernel" and a "micro-kernel." Suffice to say that a monolithic design is more common and traditional and a micro-kernel design is an improved, more elegant, more efficient—and more academically interesting—way to design a kernel. Micro-kernels have also proved much harder to produce. GNU's failure to produce a free kernel is often attributed to Stallman's decision to produce a micro-kernel. Torvalds opted for a monolithic kernel simply because he knew they could make it work. Andrew Tanenbaum, creator of the Minix operating system Torvalds drew early inspiration from, was scathing in his critique of this decision, calling it a "giant step back into the 1970's" and pronouncing Linux "obsolete" (Moody 2001:50). Torvalds' response:

> From a theoretical (and aesthetical) standpoint, linux loses. If the GNU kernel had been ready last spring, I'd not have bothered to even start my project: the fact is that it wasn't and still isn't. Linux wins heavily on the points of being available now.

A second case illustrating this "make it work first" principle was the addition of networking code to the kernel in 1992. At the time, the lead developer on the networking code was trying hard to implement a new, improved approach to networking that was taking an excessive amount of time. Because of user impatience with the lack of networking, Linus approved a separate track for networking development aiming to just "make it work first." The lead developer on this second project, British hacker Alan Cox, fast became Linus' "Number 2" and the initial networking approach was abandoned.

The networking example illustrates another key component of the social structure of Linux: Linus has final control over the kernel, but early on began delegating responsibility for specific components of the kernel to a group of trusted "lieutenants." The technical design of Linux mirrors this social structure: it is a modular design, enabling developers to work within the space of their own kernel modules that fit together to form a functioning end product. As Linux became more and more popular, the bug reports from users, and proposed patches from developers, increased in volume as well. In the first few years of Linux's life, Linus frequently released major updates just days apart, incorporating bug fixes and feature improvements submitted by contributors from around the globe and vetted by his trusted circle of lieutenants.

This social structure continues today. For example, the log for Linus' kernel tree is publicly available at git.kernel.org: you can check it out at any time, compile it, make changes to it or study it. Or, if you're not quite up to kernel hacking, you can at least get a revealing history of work on the kernel. Table 5.1 contains a snippet from the source control logs of the kernel. This is the most recent activity on Linus' kernel tree at the time of writing (Saturday, April 11, 2009, with version 2.6.30 of the kernel nearing release).

Table 5.1: Linus' Kernel Tree Log, April 11, 2009

Time	Author	Description	Commit
20 hrs ago	Linus Torvalds	Merge git://git./linux/kernel/git/dhowells/linux-2. . . [master]	d848223
22 hrs ago	Linus Torvalds	Merge git://git./linux/kernel/git/dhowells/linux-2. . .	5de4c51
23 hrs ago	David Howells	Separate out the proc- and unit-specific header directo. . .	2f2a213
24 hrs ago	David Howells	Move arch headers from include/asm-mn10300/ to arch. . .	da76166
36 hrs ago	David Howells	FRV: Move to arch/frv/include/asm/	e69cc92
36 hrs ago	David Howells	FRV: Fix indentation errors to keep git-am happy when. . .	1879346
37 hrs ago	Linus Torvalds	Merge git://git./linux/kernel/git/bart/ide-2.6	6594d0b
37 hrs ago	Linus Torvalds	Merge branch 'for-linus' of git://git./linux/kernel. . .	0534c8c
37 hrs ago	Linus Torvalds	Merge branch 'for_linus' of git://git./linux/kernel. . .	54f93b7
37 hrs ago	Masami Hiramatsu	x86: fix set_fixmap to use phys_addr_t	3b3809a
43 hrs ago	David Howells	MN10300: Kill MN10300's own profiling Kconfig	62b8e68
43 hrs ago	David Howells	FRV: Use <asm-generic/pgtable.h> in NOMMU mode	6fde836
43 hrs ago	David Howells	keys: Handle there being no fallback destination keyrin. . .	34574dd
43 hrs ago	Stoyan Gaydarov	afs: BUG to BUG_ON changes	11ff5f6
43 hrs ago	Linus Torvalds	Merge branch 'x86-fixes-for-linus' of git://git./linux. . .	e66dd19
43 hrs ago	Linus Torvalds	Merge branch 'tracing-fixes-for-linus' of git://git. . .	c2ea122

Each line in Table 5.1 represents a "commit," the name for when a modification is accepted into the code base. Of course, the specific content of each of these commits isn't important to us here, but two other things are. First, the first column shows the time of the commit relative to when I retrieved it, and, as you can, development is fast and furious, even today as Linux is a relatively stable, mature product. Second, you can see that Linus is not the author on all of these changes. In fact, if you read the description of each commit attributed to Linus, they all actually show Linus merging changes made by others into his own branch of the kernel. Other kernel contributors push changes to Linus which he accepts, rejects or modifies himself.

At the time, Linus' willingness to to delegate flew in the face of traditional theories of software development. Andrew Tanenbaum was critical of this as well:

> During the 1970s, when structured programming was introduced, Harlan Mills pointed out that the programming team should be organized like a surgical team—one surgeon and his or her assistants, not like a hog butchering team—give everybody an axe and let them chop away. . . I think coordinating 1,000 [software] prima donnas living all over the world will be as easy as herding cats. (Moody 2001:78)

In addition to the challenge in "herding cats," Linux faces a constant threat: forking. A software "fork" is one developer or group of developers takes the existing code base for a piece of software and starts their own, independent development track rather than contributing their code back to the initial project. The GPL guarantees the "freedom to fork," yet Linux has managed to avoid this fate. By incorporating the strongest candidates to lead forking projects into Linux as "lieutenants," and by consistently making progress on improving the kernel instead of chasing academic dead-ends, Linus has developed a loyal following many open source projects lack. Disagreement among BSD developers, for example, has led to several separate BSDs: NetBSD, FreeBSD, OpenBD and others.

Eric S. Raymond, who describes himself as the anthropologist of the open source movement, wrote an influential essay, *The Cathedral and the Bazaar*, contrasting the Linux development style with traditional software development:

> I believed that the most important software (operating systems and really large tools like the Emacs programming editor) needed to be built like cathedrals,

> carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time.
>
> Linus Torvalds's style of development—release early and often, delegate everything you can, be open to the point of promiscuity—came as a surprise. No quiet, reverent cathedral-building here—rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, who'd take submissions from anyone) out of which a coherent and stable system could seemingly emerge only by a succession of miracles. (Raymond 2001)

5.3 "Free Software" vs. "Open Source"

The internet was crucial to the development of Linux, which appeared on the scene just as the internet was becoming widely available to computer programmers around the globe. As the internet expanded from a relatively obscure resource for academics and the technical elite into a central component of our economy and our public culture, Linux was again in the right place at the right time. In the early days of the web, two companies competed in the web server market, Netscape and Microsoft, both offering their own proprietary systems. However, a third option, the free, open source Apache web server, would soon take off as the web server of choice, carrying Linux along with it. Linux servers were not only cheaper, but also faster, more reliable and, thanks to being open source, more flexible for the upstart web market. Virtually all of superstars of the online economy—Google, Amazon, Yahoo, eBay and more—built their businesses on Linux, the Apache web server, and a collection of other open source tools like the Sendmail email server software and scripting languages such as Perl and PHP.

However, Linux faced a credibility problem with businesses. On the one hand, IT managers ought to rejoice at a free alternative to the expensive systems available at the time. But how could the "free" software, built by a rag-tag bunch of hobbyists, possibly be as good as the software written, and supported, by a respected, successful corporation like Microsoft? Additionally, for fledgling businesses built around free software, how can one expect venture capitalists, looking to make a profit off their investments, to fund something that is "free." Of course, this is not what the "free" in "free software" means—it means "free as in speech," or *libre*, not "free as in beer," or *gratis*, but this isn't a distinction an outsider can easily pick up on right away. Complicating things was the fact that "Freeware" frequently referred to software that was gratis but not libre: software distributed for no charge, but with proprietary, secret source code.

To address some of these questions, Tim O'Reilly (founder of O'Reilly Media, a technology book publisher) organized a "Freeware Summit," assembling a who's who of free software project leaders at the time—with one notable exception: Richard Stallman, who O'Reilly felt would "disrupt the effort to achieve a consensus" (Moody 2001:167). The group decided they needed to agree upon a name if they were going to maximize their collective chance at business success, and after putting several options to a vote, "open source" was selected. The group decided that the crucial component was the sharing of source code. "Free software" got "almost no positive votes" (van Rossum 1998).

Stallman interpreted this as treason against the GNU mission: "free software" is fundamentally about an ethical, moral belief that proprietary software is a threat to human liberty and equality. "Open source" drained this of its moral content in Stallman's view, leaving only an argument about the practical benefits of sharing source code. Consequently, Torvalds has become the figurehead of "open source," claiming practical, not political, reasons drove his development of Linux—"I wanted feedback" (Torvalds 2001:194)—while Stallman believes Torvalds' de-emphases on political values undermines the cause of free software: "if you don't want to lose your freedom, you had better not follow him" (Moon 2007). When asked a question about "open source," Stallman always retorts, "I do free software. Open source is a different movement" (Williams 2002).

Despite Stallman's objections, "open source" has clearly won the war of words. In part this is by design. "Open source" was chosen to make the value proposition of free software clearer to business, but the term resonates throughout the field. For expert coders interested in the technical details of the software, it conveys the crucial bit of information: they can get at the source code and play with it. But, despite the apparent de-emphasis on morality compared to "free software," "open source" does connote moral values, but the shape these values take is ambiguous. For businesses, the transparency and "order out of chaos" nature of open source reflects the values they see in the invisible hand of the free market. For many, if not most, adherents to the "free software" philosophy, "open source" promotes a compatible, if not interchangeable, environment for building software. For non-coding end users, open source represents a kind of ethical consumption cause: rejecting software created by power-hungry corporations who want to hold you hostage by locking you into their software and by infringing on the rights of their workforce. In other words, believers in “The Liberal Dream” see a parallel between free, open markets and open source software, while critics of “The Commodified Nightmare” see open source as consistent with an inclusive, democratic culture that counteracts markets.

“Open source” floats atop the conflict in the field, allowing the market to incorporate “free” source code while maintaining rhetorical consistency with the sacred values of shared code in the context of the profane profit-seeking market. But, as Stallman correctly points out, because “open source” is about means and not ends, it becomes a shared language for actors with divergent goals to cast their actions in a morally pure, if hazy and ambiguous, light.

Having discussed the emergence of the moral field and the centrality of the concept of “open source” to the moral field since the mid-1990s, I will now turn to contemporary case studies that illustrate the moral tensions and trade-offs that developers of open source software have to negotiate. I will divide these case studies into two pairs. The first pair of cases—Google’s Android mobile operating system and the WordPress content management system—will focus on the issue of open source licensing. The second pair of cases—the Ubuntu Linux distribution and Apple’s iOS operating system and devices—will focus on the issue of usability. Both licensing and usability appear, on the surface, to be primarily about technical decisions and business decisions. However, we will see these issues are rich with moral tensions and negotiation as they are played out on the moral field of contemporary computing.

Part III

The Moral Field in Action

Chapter 6

Licensing Freedom

Chapter Preview: All software that is distributed requires a license that dictates what users of the code may or may not do. Therefore, licensing is not merely a legal decision, but a moral one, where decisions about rights & responsibilities are made. In the world of open source software, there are two classes of open source licenses: copyleft, or "viral," licenses and "liberal" licenses. The differences between these two kinds of licenses hinges on competing notions of freedom and on differing views of the market. This chapter will look at the role open source licenses, and the moral implications of these licenses, in shaping two open source projects: the Android operating system and the WordPress content management system. Android, depending on one's moral evaluation of the market and free software, is either one of the greatest open source success stories yet or a cautionary tale about overly permissive liberal licenses. WordPress, on the other hand, is an example of how the moral vision embedded in "viral" licenses can have complicated technical and economic implications for users and developers alike.

6.1 The liberal and the viral

Most code is not distributed at all, but is written by individuals or organizations for purely internal use. Think of code written by researchers to analyze some data, or a personal backup script written by a hobbyist, or internal software for inventory management, auditing, or accounting written within a particular business. When we talk about "free software" or "open source software," we are talking about software that a) is being distributed by the authors, whether for free or for a fee, b) with the source code included, and c) with a license that grants the licensee certain rights and responsibilities with respect to accessing, modifying, and redistributing the source code for that software. Licenses are moral. Both in the sense that they prescribe the right and wrong uses of software, but also in that moral reasons are provided to justify the decision to open source software. Beliefs about the moral virtues, or moral shortcomings, of the market and private enterprise inform people's views about why open source is a good strategy. Some view open source as a means to protect software from the negative effects of market, while others view open source as a strategy for market success. The clearest way to see this is to look at an ongoing debate between different varieties of open source licenses.

When releasing code under an open source license, there are two kinds of licenses from which to choose. The first group consists of "copyleft" licenses (such as the aforementioned GPL), and the second group of open source licenses are frequently referred to as "liberal" licenses (such as the BSD, MIT, and Apache licenses). The GPL was designed to protect what Richard Stallman dubbed the "four freedoms" that any computer user should have with free software. Counting, like a computer programmer, from zero (Free Software Foundation 2009):

0. The freedom to run the program, for any purpose.
1. The freedom to study how the program works, and adapt it to your needs.
2. The freedom to redistribute copies so you can help your neighbor.
3. The freedom to improve the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits.

The primary difference between the GPL and the liberal licenses are the conditions placed on the fourth freedom. The GPL requires that any redistributed modifications to a program preserve these same four freedoms for their users. In other words, the GPL is "viral": building on GPL'd code means your code will have to be licensed under the GPL as well. You cannot take GPL code and incorporate it into proprietary software. Stallman dubbed this technique of protecting the rights of the user, not the producer, the "copyleft."

On the other hand, the liberal licenses place less onerous conditions on derivative code: the author must give credit to the original author in some fashion, but they are permitted, if they choose, to distribute their modified code under a proprietary, closed source license.

This relatively minor difference in open source licenses has some major practical and moral implications. Practically-speaking, the two licenses encourage different use cases for software. Without a "viral" clause, liberal licenses allow proprietary software vendors to include and modify open source code in their projects without the need to redistribute their changes, a "freedom" that can enable competitive advantages (rivals can't use their modified code in their own projects) but also avoid perceived headaches associated with GPL code: the need to participate in a larger community, publicly support any released code, and the fear of being sued for violating the terms of the GPL.

Which license is more "free"? This question is a source of much debate among open source developers and businesses. On the one hand, the liberal licenses are more permissive with respect to developers who are modifying the code. For instance, Matt Asay (2009), an open source industry veteran who has worked for open source companies Canonical and Alfresco, criticizes the GPL for it's lack of inclusiveness, scaring potential contributors away because of the limitations it places on redistribution:

> The GPL is like opening a cannister of radioactive waste: while your competitors can touch it, you're dead certain that they won't.
>
> Given that openness is increasingly a winning business model. . . one has to wonder if pretending to be open through the GPL accomplishes as much as fully opening up through Apache-style licensing would.

Open source advocate Eric Raymond makes "The Economic Case Against the GPL" (Raymond 2009):

> If we live in a "Type A" universe where closed source is more efficient, markets will eventually punish people who take closed source code open. Markets will correspondingly reward people who take open source closed. In this kind of universe, open source is doomed; the GPL will be subverted or routed around by efficiency-seeking investors as surely as water flows downhill.
>
> If we live in a "Type B" universe where open source is more efficient, markets will eventually punish people who take open source code closed. Markets will correspondingly reward people who take closed source open. In such a universe

> closed source is its own punishment; open source will capture ever-larger swathes of industry as investors chase efficiency gains.
>
> ...
>
> I think we live in a type B universe – that is, one in which the GPL is unnecessary rather than futile. Mind you, I am not claiming the GPL is entirely useless. It's a signaling behavior, like wearing a crucifix or yarmulke or pentagram – it helps build trust groups. But it has costs, too — it creates a lot of needless fear from potential allies and users who suspect they won't be able to control their exposure if they let it in.

In addition to this economic libertarian take, others make more of a ethical libertarian argument in favor of liberal license. For instance, Jacob Kaplan-Moss (2008), co-founder of Django, an open source web framework, likens the decision to use the liberal BSD license instead of the GPL to a defense of free speech:

> "Freedom 0" is the freedom to use software for any purpose whatsoever.
>
> I've always been confused on this point: the FSF itself coined this term, but their flagship license, the GPL, prohibits certain forms of reuse. That's always rubbed me the wrong way. The BSD, on the other hand, is about as close to Freedom 0 as you can get.
>
> We felt strongly on this point: we really wanted Django to be used as widely as possible... and "anyone" means "anyone." This goes along the same lines as Voltaire's famous line: "I do not agree with a word you say, but I will fight to the death for your right to say it." We felt the same should apply to software: I may not like what you do with my software, but you should have the right to do it.

Each of these positions embodies a moral stance toward the market. Raymond has faith that markets will reward openness so coercive enforcement is unnecessary and even harmful, while Kaplan-Moss implies that whether or not Raymond is correct about markets promoting openness, it is wrong, in principal, to place restrictions on what people can do with open source code in the market—including restrictions that prohibit them from removing it from the open source commons.

On the other hand, proponents of a copyleft license, argue that the GPL better preserves freedom for future generations of developers and users. Stallman criticizes liberal licenses as a way to choose short-term gain over long-term preservation of freedom for users. For instance, the X Windows system, the software that allows Unix systems to draw graphical windows on the screen like Microsoft Windows or Apple's operating systems, was developed at MIT and released under the MIT license, which, like the BSD license, is extremely permissive. The result was X Windows being incorporated into proprietary Unixes without the source being made available. Stallman argues:

> The developers of the X Window System did not consider this a problem—they expected and intended this to happen. Their goal was not freedom, just "success," defined as "having many users." They did not care whether these users had freedom, only that they should be numerous (Stallman 1999).

Matt Mullenweg, co-founder of the GPL-licensed content management system WordPress, calls the GPL a "bill of rights for software" (Pick 2000):

> I feel it's the most moral of all the licenses. The GPL has a clause which basically says that if you build something on top of a GPL product, it must also be GPL. So the freedoms are maintained, even in derivatives or things built on top of it. That's why all plugins and themes for WordPress are GPL. This engenders an incredible amount of creativity because it allows things to build on what came before, versus other licenses where a lot of the innovation and development might happen in something that is essentially closed source. It's based on open source code, but there's no reason for anyone to release it, so essentially all that development goes into a sort of innovation black hole.

Just as critics of the GPL had both an "economic" and "principled" component to their argument, Mullenweg's position is both about the morality of the GPL, but also beliefs about the way the GPL and liberal licenses cause different market outcomes: avoiding "innovation black holes," "engendering creativity," etc. In other words, the "practical" factors involved in choosing an open source license is closely interwoven with the "moral" factors, and certain features of market competition, or protection from market competition, are branded as "engendering creativity," on the one hand, or enabling efficiency on the other.

This division between interpretations of open source licenses and freedom can be seen throughout many open source communities. In this chapter, I will look at two case studies. First, the Android mobile operating system. Second, Mullenweg's WordPress content management system.

6.2 Android's definition of open

Android is an open source mobile operating system produced by Google. Google produces Android internally and releases it under an open source license which permits handset makers (such as Motorola, Samsung, and HTC) to take it for free and put it on the smartphones they build and sell. Android's licensing is somewhat complicated though. Android uses the Linux kernel, which is licensed under the GPL. However, the software frameworks and toolkits that make Android distinctive—the parts developed by Google that Android programmers and users interact with—are licensed under the Apache Software License, a liberal license which imposes no "copyleft" restriction on development. Additionally, on top of that layer, a set of proprietary Google applications—such as Gmail, Google Maps, and the Android Market—are licensed to specific vendors wishing to brand their phone "Google Experience" phones.

At the time I am writing this, Google is claiming that 350,000 new Android devices are sold each day. Android, it appears, is the "Linux for the masses" that free software devotees have been dreaming of for decades now. Additionally, Google's marketing of Android to developers and users alike has emphasized the "openness" of the platform, particularly in comparison with Apple's proprietary iOS and tightly controlled App Store. For example, at their 2010 developers conference, Google I/O, Google Vice President Vic Gundotra, against a "1984" backdrop evoking Apple's famous Macintosh ad, dramatized Google's introduction of Android:

> If Google did not act, we faced a Draconian future, a future where one man, one company, one device, one carrier would be our only choice. That's a future we don't want.

Beyond just Android, Google's long-standing company motto, "Don't be Evil", represents the moralizing stance Google adopts with respect to it's competition. However, despite it's rhetoric, Google has seen harsh criticism from developers for deviations from the open ideal.

Unlike the Linux kernel, development of Android occurs almost exclusively inside Google. Day-to-day development branches of the source code are not publicly available—instead, Google releases "code dumps" of Android major releases. In 2011, market research firm VisionMobile released a report on what they call the "Open Governance Index." They evaluated how open a project is according to four criteria: access to source code, transparency of the development process, permissiveness towards derivatives, and structure of the community (VisionMobile 2011:p. 14). Of the open source projects they looked at, they ranked Android last according to each of these criteria, earning a 23% overall score on their Open Governance Index. This "open source but closed development" model has drawn criticism from open source developers such as Joe Hewitt (2010):

> I cut my teeth in the software industry working on the Mozilla open source project, so when I hear others talk about openness, but see them omitting important facets like a public source tree and outsider commit privileges, my bullshit radar goes off.

Android received particularly heavy open source scrutiny in Spring 2011 when Google announced that Android 3.0, nicknamed Honeycomb, would never see open source release. Honeycomb began shipping installed on Motorola's XOOM tablet in early 2011, but in March, Andy Rubin, head of the Android group at Google, announced there would be no open source release of Honeycomb because Google "took a shortcut" in Honeycomb's development and the source code wasn't ready for release yet.[1] The "shortcut" comment provoked widespread criticism, even ridicule. A few months earlier, in response to criticism that Android wasn't truly "open", Rubin had responded with a tweet[2]:

> the definition of open: "mkdir android ; cd android ; repo init -u git://android.git.kernel.org/platform/manifest.git ; repo sync ; make"

If you don't understand that, you're not alone. That's the code for cloning the source code of Android from Google's code repository. As I discuss further in the chapter on usability, this definition of openness extends to one group of users: programmers. Six months later, Rubin was explaining how the lack of a `repo sync` for Honeycomb did not mean an end to Android's

[1]http://www.businessweek.com/technology/content/mar2011/tc20110324_269784.htm
[2]https://twitter.com/Arubin/status/27808662429

openness, but even Android fans were growing restless. On the blog *Android Central*, Jerry Hildenbrand, signing his post as " A heartbroken Android evangelist" responded to Rubin[3]:

> Andy Rubin goes on to tell Bloomberg *"Android is an open-source project. We have not changed our strategy."* You could have fooled me, Andy.

In the comments thread on the post, many echoed Hildenbrand's sentiment. For instance, "GlueFactoryBJJ":

> The real fear is that, as has been noted, that Google is misinterpreting the reasons for the success of Android phones. Techies are the people who started promoting Android because of its openness to being "customized" vs. the locked down system in iOS (iPhone/iPad, no launchers, no live backgrounds, no widgets, no keyboards, etc. because they affect the "iOS experience"). We recommend products to our friends. I, personally, have been directly responsible for at least 10 Android phone/tablet purchases (vs. Blackberry or iPhone). And these people have told their friends, etc. IMO, THAT is what is driving Android's current success.
>
> Our family has 4 Android devices (DroidX, 2x Fascinate, Nook Color) and will add one more as soon as there is a choice of dual core phones on Verizon. I got them BECAUSE of their openness, NOT because of the "experience". BECAUSE of their openness, I can create/refine the experience *I* want through Launcher Pro and other apps. The same with my family. They can make it work the way that works best for THEM. Not be limited to what Steve Jobs and Co. think things should work (yes, their stuff does work well, but it isn't MINE).
>
> ...
>
> As Google has gotten bigger and more powerful, they are forgetting their roots AND the reason they have been successful. They are making a good bit of money and having a lot of success because of their openness. I hope they don't go down that path and let "absolute power corrupt absolutely".

"DaEXfactoR" agrees:

[3]http://www.androidcentral.com/google-not-open-sourcing-honeycomb-says-bloomberg

> Mark my words, this is only the beginning. It seems they want to follow in the path of a certain crazily popular tablet that begins with an i. I believe this is how they think they can duplicate that success, by controlling the "experience". We have accepted the long over due updates on Android. We have endured the wild wild west feeling of the Android Marketplace. We deal with these things because we love Android and it's openness. So we are willing to live with the risks and inconsistencies that come with an open platform.

Much of Google's flexibility with respect to Android being "open" or not is a product of Android's liberal license. Android's choice of the Apache license means that handset makers can take Android and make proprietary modifications to the operating system they install on their phones. Google's "Open Handset Alliance" FAQ makes it clear this is an intended consequence of this licensing decision:

> Apache is a commercial-friendly open source license. The Apache license allows manufacturers and mobile operators to innovate using the platform without the requirement to contribute those innovations back to the open source community. Because these innovations and differentiated features can be kept proprietary, manufacturers and mobile operators are protected from the "viral infection" problem often associated with other licenses.
>
> [...]
>
> Because the Apache license does not have a copyleft clause, industry players can add proprietary functionality to their products based on Android without needing to contribute anything back to the platform.

Consequently, Android's "openness" is a highly qualified variety: while it is released as open source software, it is not fully developed "in the open," and in the vast majority of cases, someone who purchases an Android phone cannot obtain the source code for *their* phone.[4] One can go to source.android.com and download the source code released by Google, but each of the major handset makers selling Android phones have developed their own custom, proprietary modifications of Android that they bundle with their phones. Additionally, manufacturers include hardware-level protection against installing "unauthorized" copies of

[4]Google does sell a "Nexus" series of phones that run stock Android, but they represent a tiny fraction of Android handsets sold.

the operating system on the phone. Motorola has been particularly aggressive about this. For example, their "Droid X" phone, released Summer 2010, ships with a locked bootloader that prevents the phone from booting if an unauthorized, non-Motorola version of Android is installed.

This is ostensibly a security measure, but in effect it means that even if you want to install pure open source Android on your phone, you cannot do so without using fairly arcane, risky, and arguably illegal techniques to break these protections. For example, hackers eventually found a way to circumvent the locked bootloader on the Droid X and install custom open source builds of Android, but the procedure is not for the faint of heart. Additionally, once one breaks through these barriers and installs open source Android, frequently the drivers necessary to enable all of the phone's hardware—camera, GPS, WiFi, Bluetooth, etc.—are closed source. Given the realities of what it takes to enjoy Android's "openness", the platform ends up looking not that much different than Apple's iOS devices: iPhone users can "jailbreak" their iPhones, gain root access to their phone's operating system, and install apps from outside Apple's App Store. Yes, this violates Apple's End User License Agreement and voids your warranty—though the same applies, in practice, to the vast majority of Android phones.

It's also important to recognize that much of what people think of as Android—Google Maps, Gmail, Google Calendar, the Android Market—are neither GPL or Apache-licensed, but are proprietary Google applications. Google is also rather restrictive about the use of the Android trademark: manufacturers that have taken Android open source code and released devices without getting Google's approval have been unable to use the "Android" label for their devices. In May 2011, Google's contracts with both Motorola and Samsung were made available by the court during an ongoing lawsuit by Skyhook Wireless against Google. The gist of the lawsuit is Google's insistence on bundling their own location data services with the phone instead of allowing Samsung to substitue Skyhook Wireless' competing service (Patel 2011). In other words, one of the conditions of Google's licensing the Google Apps and the Android trademark appears to be that the phone use Google's location service to collect data about your whereabouts.

In short, Android is open—in some respects, for some people. Android is open for developers willing and able to purchase Google's Nexus series of "developer phones," but not for end users who purchase an HTC or Samsung phone at their local Verizon store. Android is open for phone manufacturers willing to strike business deals with Google to offer "Google

Experience" phones, allowing them to sell phones with Google's applications and gain early access to development versions of Android that are not yet publicly available. While marketing Android as the open alternative to Apple's iPhone, the characteristics of its openness firmly position Android in the experts/markets corner of the moral field, being open primarily for elite developers and Google's partner corporations.

Contrast this qualified openness with a more pure open source project such as the Linux kernel. As I did with the Linux kernel (see page 55), anyone can go to git.kernel.org and view the up-to-the-minute state of the Linux kernel while it's undergoing development. Any business or private individual has this level of access. The Linux kernel is also licensed under the GPL license, which prohibits you from distributing proprietary modifications. Because Android runs on the Linux kernel, when HTC or Motorola make kernel modifications to allow their hardware to work with Android, they have to release these changes.[5]

Defenders of Google's approach with Android, however, argue that the idea of a purely open, "viral" license is a non-starter given the realities of the mobile industry. Had Google adopted the GPL for Android, many of the manufacturers and carriers currently embracing the platform may have been resistant to the requirement that they open source their modifications to Android—which they feel differentiate them from competing Android vendors in the market.

On the other hand, by not requiring HTC, for example, to give back to the open source community, the liberal open source license enables manufacturers to free-ride on the hard work of the open source community, giving nothing back in exchange for free code, violating the hacker ethic.

Android protects the freedom of carriers and handset makers, but will it cultivate an environment of free, open, hacker-friendly technology for the larger public? In terms of the moral field, is the freedom of the programmers working in the mobile industry (the top-left quadrant) what open source ought to protect, or is software freedom for the community and the public (bottom-right) the goal?

The split between the liberal and viral licenses points to the difficulty of integrating a shared community object into the market. Is open source code a *gift*, released by a software author with no conditions, or is open source code a *public good*, released under a pact between coders

[5]But not all changes to the operating system: recall different levels of the Android system have different licenses and the Android layer is not GPL. This is confusing.

to "share and share alike." Each of these enable the sharing of code, but the two models can encompass dramatically different relationships and obligations between producers and users. For the skeptic of the market, the "public good" model offers the greatest protection from the corrupting influence of commerce. For the believer in the virtues of the market—efficiency, wealth creation, progress—making code as amenable to market adoption as possible offers the best chance for making a difference with code. Proponents of this view point to the sales numbers of Android to support this view. Despite the trade-offs Google had to make to get handset makers and carriers on-board with using open source software, the long-term future of open source software is brighter because of it, the argument goes.

6.3 The GPL and the Web

While mobile operating systems such as Android and iOS are the hot new thing today, they actually represent a retreat to a more traditional relationship between a platform and its applications. The "app store" model pioneered by Apple and now copied by other mobile platforms may be a departure from the way applications are distributed on traditional desktop operating systems, but the idea of consumers purchasing licensed copies of an application to run on a local device is fundamentally the same. The web, however, represents a more radical shift in software distribution, licensing, and ownership. Does a user of Facebook, for example, *install* Facebook? In a sense, yes: everytime you load facebook.com in your browser, you are downloading the HTML, CSS, and Javascript files that collectively present you with Facebook. However, these files on their own are not so useful if the user can't connect to the *real* Facebook, which lives on servers in Facebook's datacenters. No one other than Facebook has installed this software. There is no software license for Facebook because it has never been distributed.[6]

This model is referred to as "Software as a Service" (SaaS). Instead of selling copies of software, companies host the software on their own web servers and sell access—through a web browser or API—to interact with the application. Switching from Microsoft Office to Google Docs doesn't require installing anything new on your computer: you just start accessing Google Docs on Google's servers through your web browser. The vast majority of

[6]There are, of course, end user license agreements that set the terms for what a user can and cannot do on the service, but this is different than licensing the software itself.

experience that ordinary people have with web applications are of this SaaS, hosted variety.

But what role does open source play in this new world of hosted web services replacing locally installed applications? A huge role, actually. We're in a golden age of open source web technologies. Facebook was built upon a traditional open source "LAMP Stack": the Linux operating system, running the Apache web server, storing data in a MySQL database, with code written in the PHP programming language. Each of these components is open source, and as Facebook has expanded and outgrown their initial architecture, they've even continued to open source new technology as they've developed it, such as a new database (Cassandra) and a tool that transforms their PHP code into faster C++ code (HipHop for PHP). A similar story can be told for many, if not most, of the largest websites going today: Google, Amazon, Twitter, Wikipedia, and so on.

Of course, in each case, the *tools* are open sourced, not the web application itself. It is a golden age of open source for web *programmers*, not necessarily for web users. To the extent that web users can become web programmers and web server administrators, it is easy for them to control their own software. For instance, one can run their own web server and install an open source content management system, such as WordPress. They will then have complete control over their data and the source code running their applications. There are obviously constraints on the ability of ordinary users to do this, but WordPress is a good example here as it is perhaps the most accessible open source web application for non-technical users. For a few bucks per month, one can get cheap shared web hosting and install WordPress. On the other hand, most users of the web do not know what "shared web hosting" means, let alone how to procure and set up "cheap" shared web hosting.

6.3.1 WordPress

With this strange state of open source on the web, a project like WordPress is particularly interesting. WordPress is software you install on a web server to run your own blog. WordPress is, like Facebook, built on the LAMP stack: you can run it on a Linux server using all open source components. WordPress itself, however, is free and open source software, licensed under the GPL. You can download the software from wordpress.org for free and have complete access to the source code.[7] WordPress has a "plugin" architecture allowing developers to write themes and plugins that extend one's WordPress installation. The project's website hosts a

[7]http://wordpress.org/latest.zip

directory that, as I write this, consists of 15,444 plugins and 1,414 themes, all also licensed under the GPL. This is a lot of code, contributed by a lot of developers.

However, WordPress has a split existence: WordPress.*org* is home to the open source project, and WordPress.*com* is a hosted service run by Automattic, Inc. (that's not a typo: the company is named after founder Matt Mullenweg). In addition to WordPress.com, Automattic hosts several other popular web services, such as PollDaddy (online polls), Akismet (a spam detection service), and Gravatar (one of the services responsible for showing the same avatar by your name whenever you post comments around the web). The code for these services is not open source. It's not licensed at all because it's not distributed.

According to Automattic, there are over 52 million WordPress blogs in the world and Wordpress.com hosts about half of them.[8] In other words, half of all WordPress installations are "open soure" installations, while the other half are hosted by Automattic. The software running WordPress.com and what is available on WordPress.org are not dramatically different, and features added to the hosted service frequently make their way into the open source release. The story behind how one code base can support both a "software as a service" business and also support a traditional free, open source project tells us much about the state of open source on the web today.

In 2003, the big kid on the blogging software block was Movable Type. Movable Type is software you install on a web server that allows you to write, edit and publish a blog from within a web browser. For the first generation of bloggers, Movable Type was the go-to choice if you wanted to host your own professional-quality blog. It was free—but free as in beer, not free as in speech. In other words, it was not open source software, even though it was distributed as source code because it was written in an open source programming language (Perl). Users did not have the right to distribute modifications to Movable Type or to fork the software and take it in a new direction. But for many, as Mark Pilgrim (2004) explains in an oft-cited blog post on Movable Type, it was "free enough":

> Movable Type came with source code, so hack-minded individuals could add features like threaded comments or IP throttling. You weren't free to redistribute complete versions of your hacked copy of Movable Type, but you could release patches, and that was free enough. It had a well-designed plug-in architecture, so it

[8]http://en.wordpress.com/stats/

> could be customized with a dizzying array of third-party plug-ins. You weren't free to redistribute Movable Type with those plug-ins pre-installed, but you could install the plug-ins separately, and that was free enough.

However, two things happened in 2004 that changed Movable Type's dominance of the self-hosted blogging software market. First, the company behind Movable Type (Six Apart) changed Movable Type's licensing and began charging for the software. Second, an upstart challenger to Movable Type emerged, the GPL-licensed WordPress. Movable Type's mistake helped give WordPress enough extra momentum to push bloggers like Pilgrim to WordPress. (A few short years later, in 2007, WordPress was the dominant blogging platform and Six Apart reversed course and switched Movable Type's license to the GPL, but it has never recovered in popularity.)

In 2003, WordPress began as a fork of the b2 blogging software. The original developer of b2 had stopped developing the software, and two b2 users (Matt Mullenweg and Mike Little) took the b2 source code and started WordPress. b2 was GPL-licensed, so they were completely within their rights to do so—in fact, the original b2 developer would later become a WordPress contributor. WordPress specifically focused on usability and making it as easy as possible to install WordPress without being a programmer, advertising a "5-Minute Install."

Like Movable Type, WordPress allowed programmers to write third-party plugins to modify and extend WordPress' behavior. For developers who want to build a website that isn't quite what WordPress is designed to do out of the box, WordPress is still an option because it's relatively easy to take advantage of all of the common features all websites share that WordPress does provide (user management and authentication, a database for storing information, a theming system for styling publicly viewable pages, etc.) and just write a plugin that adds the extra bit of functionality you need. In other words, it's easier to take tools that already work and modifying them for your purposes than it is to build your own tools from scratch. Unlike Movable Type though, WordPress was GPL-licensed, so developers knew they could trust WordPress. Mark Pilgrim again:

> WordPress is Free Software. Its rules will never change. In the event that the WordPress community disbands and development stops, a new community can form around the orphaned code. It's happened once already. In the extremely unlikely event that every single contributor (including every contributor to the

> original b2) agrees to relicense the code under a more restrictive license, I can still fork the current GPL-licensed code and start a new community around it. There is always a path forward. There are no dead ends.

In the 2000's blogging went from being niche hobby of early adopters to a mainstream publishing vehicle. Today virtually every major news outlet have blogs that either supplement or replace traditional publishing mediums. The *New York Times*, CNN, *Wall Street Journal*, and FOX News, to name a few, run their blogs on WordPress.[9] Government officials have blogs. (Nancy Pelosi, Benjamin Netanyahu, and Jim DeMint have WordPress blogs. The White House blogs opt for Drupal, another open source platform.) This growth of blogging has been great for web developers, and many new web developers joined the field who were not already programmers. WordPress founder Matt Mullenweg, for example, was not a professional programmer when he started WordPress: he was a 19-year-old political science major at the University of Houston. He dropped out, and in 2008 was number 18 on Inc magazines list of top 30 entrepreneurs under 30 (Inc. 2008).

To summarize, a) the GPL was crucial to WordPress' early success, b) web development attracted a new generation of programmers. Against this backdrop, it may be surprising to find that the GPL has proved a consistent source of conflict within the WordPress community. The conflicts tend to revolve around two issues: first, WordPress' policy, and legal assertion, that WordPress plugins and themes must be GPL-licensed, and second, the fact that Automattic has raised tens of millions of dollars in startup funding while building a business model around closed-source services, such as the anti-spam service Akismet, that are tightly integrated with WordPress.

WordPress is licensed under Version 2 of the GPL, which was released in June 1991.[10] Modern web applications did not exist and were not a concern at the time this license was drafted. Consequently, there are some grey areas when it comes to interpreting what the GPL entails for web applications. Without getting too far into the technical details: web applications today are written in "dynamic" programming languages. This means they are not compiled in advance like traditional applications: code is actually compiled *at runtime*. So whenever you type a URL into your browser and request a page from a stock WordPress blog, the source

[9]Not to mention The Society Pages and *Contexts* Magazine. :)

[10]GPL Version 3 was released in 2007. Version 3 makes some rather controversial changes I'll return to in a bit. For now, I'll just emphasize that many projects, such as the Linux kernel, remain on Version 2, so WordPress is not at all unusual in this respect.

code for WordPress is loaded into memory on the server and a response is compiled and sent back to your browser, just for you.[11] With that in mind, here's the scenario that has caused some controversy in the WordPress community: say you're writing a plugin or theme for WordPress, which entails creating some files of your own, filling them with PHP code, and putting them in a special directory that WordPress loads at runtime. Is distribution of your code subject to the "viral" clause of the GPL? Must it be GPL as well? On the one hand, you are not modifying the files containing WordPress code at all. You're only adding files. Why would WordPress copyright apply at all? The Free Software Foundation, authors of the GPL, believe it does apply: because your code is loaded into memory with WordPress at runtime, it *does* modify WordPress (Mullenweg 2009b, Jaquith 2010). Is WordPress the bits stored on disk as files filled with php code? Or is WordPress the bits manipulated in memory and executed by the CPU every time it runs?

Nailing down the ontology of a web application obviously entails answering some technical questions, but as we've seen throughout this paper, they are not *just* technical. There are legal issue as we're discussing how a license applies to a particular use of a piece of software, but these questions become moral questions as well. To see how, let's look at some specific situations.

As I mentioned above, WordPress hosts a theme and plugin directory where one can download thousands of plugins or themes. WordPress takes the position of the Free Software Foundation on plugin and theme licensing: they must be licensed under the GPL. Therefore, only GPL-licensed plugins and themes are allowed on the directory. The directory is the primary way WordPress users find new themes and plugins to install, in large part because searching and installing the wordpress.org directory is integrated right into WordPress itself. In 2008, WordPress removed 200 themes (nearly 30% of the directory) that failed to comply with the GPL or with additional GPL-related policies in the WordPress directory. (For instance, theme authors are not allowed to include links back to websites that sell non-GPL software.) This action prompted criticism not only from those kicked out of the directory, but from other developers and users of WordPress who opposed what they view as the draconian aspects of GPL enforcement, but also an alleged hypocrisy and conflict of interest between WordPress.org and the commercial effort, Automattic, supporting WordPress.

For instance, blogger Duncan Riley criticized Mullenweg for his "jihad" against WordPress themes (Riley 2008):

[11]Just as an aside: isn't that amazing? This is how fast computers are today.

> Mullenweg has an issue with people profiting from WordPress despite the fact that he's the one person who profits from it the most. It's all very well and good wanting to be a open source purist, but you don't get to play legitimate purist when you're making money from it yourself.

One of the developers with themes booted from the directory, Spectacu.la, made similar allegations:

> I'm not quite sure why WordPress.org has such an issue with commercialism - Automattic, who run WordPress, are a very commercial entity. The $29.5m of funding behind them is hardly insignificant. $29.5m is about 29,500,000 times more profit than we've made from Spectacu.la so far. We don't have the luxury of venture capital either, being funded entirely out of our own pockets.
>
> The more I think about it, the more I realise that Automattic appear to want it every way - to have their own protected IP and commercial interests whilst keeping all others as far away as possible.

There are two issues here: the GPL and commercialization, and whether or not the WordPress-related proprietary services Automattic runs violate the GPL just as non-GPL themes and plugins do. First, let's look at Akismet, an anti-spam service. Akismet is proprietary code. It runs on Automattic servers and is not distributed. To use Akismet on a WordPress blog, you install a WordPress plugin that connects to the Akismet service which then checks every new comment on your blog for spam. Automattic makes money from Akismet by charging for spam protection on high volume, commercial websites. The WordPress plugin itself is GPL licensed. The external service that it connects to is not. Automattic follows this model for several of its services. For example, it also runs a WordPress backup service, VaultPress, that operates in a similar way: a GPL plugin connects to a proprietary service and, for a fee, makes backups of your site's data.

In other words, Automattic builds services around the WordPress open source project. The code Automattic contributes to WordPress—many, if not most, of the core developers of WordPress are employed or associated with Automattic in some way—as well as to the plugins that interact with their services, is GPL-licensed. The services, however, are not open source. They consist of proprietary, non-distributed code, and Automattic charges to access these services. In other words, the “Software as a service” model introduced above. These services are built, presumably, with the same open source tools that build WordPress—the LAMP stack, more or less—but unlike WordPress, these products are not distributed.

The GPL does *not* prohibit charging for software. It merely prevents the seller from prohibiting their customer from taking the source code, making modifications, and redistributing it. For example, several “Premium Theme” companies, such as StudioPress, charge a fee for downloading their GPL-licensed theme. This comes with a risk that some people will take the theme and distribute it for free somewhere else, but StudioPress’ Brian Gardner claims licensing their themes under the GPL did not hurt sales (Cook 2010a). Paying customers get support from StudioPress, as well as immediate access to new versions of the software, which is important in the fast-paced world of web development where new features and security patches are frequent.

In 2010, the outspoken author of the popular Thesis theme, Chris Pearson, made a very public refusal of adopting the GPL for Thesis (Cook 2010b):

> ...the GPL basically stipulates that anyone can take your code and do whatever they want with it. . . legally. In other words, the GPL is legal castration for anyone who is trying to run a business that is in any way reliant upon unique, innovative code. . . . I am (and have been since 2006) a leading presence in the WordPress theme development community. My work has always received a lot of attention and scrutiny, and that has made it a prime target for copycats and people who are out to make a buck as easily as possible. In situations like this, the GPL gives all the power to the “little guy” while robbing the true innovators of any protection whatsoever. I’m an innovator, my work is creative, and the GPL is NOT on my side here; this is strike two for the GPL.

After appearing as a guest on the Mixergy podcast with Matt Mullenweg and more or less saying “sue me” to Mullenweg (Warner 2010), Pearson eventually caved and released Thesis under a GPL license.[12] The case brought a lot of attention to the issue, however, sparking much discussion about the politics of the GPL. Support of Pearson ranged from those declaring the GPL “Marxist” (Hangen 2010) to the more frequest claim that, among open source licenses,

[12]Pearson to Mullenweg: “You’d be better off saying nothing and trying to take me to court.”

the GPL is "less free" than the liberal licenses. For example, developer of the popular Mac blogging client MarsEdit, Daniel Jalkut (2009) says of WordPress:

> All of its developers have something to be immensely proud of. But whenever I am reminded that WordPress is GPL, my passion for it takes a bit of a dive. I'm more comfortable with the true freedom of liberally-licensed products. If a liberally-licensed blog system of equal quality, ease of use, and popularity should appear, my loyalties to WordPress would not last long.

This sentiment—that the liberal licenses better represent "freedom in it's purest form" (Rutkowski 2010)—is a commonly heard critique of the GPL. After all, the GPL makes more restrictions on what programmers can do with code, therefore, by definition, this means less freedom, right? Advocates of the GPL argue that the restrictions of the GPL better protect freedom in the big picture though: ensuring improved code remains free and open for future generations of programmers and users and promoting a stronger overall ecology of free software.

Mullenweg (2009a) replies to Jalkut:

> Mr Jalkut conflates what he perceives as his freedom as a developer with freedom from a user's point of view. The things the GPL "takes away" from him, like being able to license his derivatives under a more restrictive license, are in fact protecting the freedoms of the users of his code. That's who the GPL was written for. From the Free Software Definition:
>
>> Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it refers to four kinds of freedom, for the users of the software [...]
>
> It's user freedom that the GPL was created to protect, just like the Bill of Rights was created to protect the people, not the President. The GPL introduces checks and balances into an incredibly imbalanced power dynamic, that between a developer and his/her product's users. The only thing the GPL says you can't do is take away the rights of your users in your work or something derived from a GPL project, that the user rights are unalienable. You are free to do pretty much whatever you want as long as it does not infringe on the freedoms of others. (Sound familiar?)

For scholars of libertarian versus communitarian conceptions of freedom, this debate will sound quite familiar. Proponents of the “The Liberal Dream” see a free market with minimal restrictions on individuals as key to maximizing freedom (Fourcade and Healy 2007), and at an individual level, liberal open source licenses are less restrictive for any given developer who takes a chunk of open source code and builds something with it. A handset manufacturer such as Samsung, for example, can take Android’s liberally licensed code, make proprietary modifications that they feel will distinguish their products in the market, and sell their phones with these features without fear that competitors will be able to freely poach these features for their own phones. A WordPress theme or plugin developer can ensure that any users of the code they’ve written will be paying them, the author, to use their code rather than some other 3rd party coming along and taking away their relationship with their customers and their income.

On the other hand, adherents of “The Commodified Nightmare” vision of markets will see flaws in this argument. Open sourced code represents a commons that all developers can drawn from and contribute back to. If it’s permissible to freely take from the commons but close off your own products, that turns what was collaboration into zero-sum competition which will, over time, weaken the commons, promoting proprietary, walled gardens over a cooperative ecosystem of interlocking parts.

Which vision is ultimately more correct is not at issue here. What is of interest is how the deliberation between the two shapes one of the core debates in open source software development: what counts as open source? Licensing is not a simple legal decision, but one that entails grappling with questions about fairness, community, and responsibility, in addition to questions of self-interest and economics.

Chapter 7

Usable Freedom

Chapter Preview: What good is open source if only programmers can benefit? Hackers have theorized that open technology is a necessity for a democratic society, but open source projects have struggled with reaching the mainstream consumer market. This is frequently attributed to usability shortcomings. Solving the usability problem, however, can put open source projects in tension with their ideals and their core constituencies. This chapter will look at two case studies—Apple and Ubuntu—to dissect the way actors engage the moral field in the pursuit of making computers more usable. Usability may appear to be a politically neutral goal at first glance, but questions about how to best build usable software, and what audience consists of the baseline for usability, leads developers, project leaders, and companies to engage the moral field to advance their goals.

This chapter will focus on two case studies, Apple and Ubuntu. In the standard open source typology, Ubuntu is "open" while Apple is "closed." However, as I have argued, the moral tone of these terms actually mask underlying moral and political tensions. Making software "usable" requires negotiating issues of commercial viability versus community ideals, answering questions about whose usability needs count in software design, and making decisions about what sort of organizational processes produce software that is "usable."

I will start by looking at Apple, whose success in recent years has made them the poster child for usability, on the one hand, and for ruthless capitalism on the other. Apple is, it seems, either the best thing to ever happen to computing or the new "Evil Empire" (a nickname given to Microsoft during it's heyday). The realization that I had to include Apple in this discussion was a turning point in my analysis. Initially, I viewed this project as focused solely on open source software and the communities that produce and use it. However, this focus proved too narrow. As we saw in Part II, free and open source software are only relatively recent manifestations of a broader culture involving the development of technology. And as we will see in this chapter, Apple is a key player in the discourse of "open" as well. Its products, and the process by which they are created and distributed, are the focus of much moral energy by developers and others in the industry.

Next, I'll turn to Ubuntu, a Linux distribution that has gone to great lengths in recent years to improve the usability and mainstream appeal of Linux as a desktop operating system, frequently citing Apple as both a role model and competitor in the process. Unlike Apple, however, Ubuntu is an open source product and struggles to balance the interests and ideologies of its open source community with its goals of mainstream market success.

7.1 Apple on the Moral Field

On January 27th 2010, Apple cofounder Steve Jobs introduced Apple's latest much-hyped device: the iPad, a 9.5x7.5 inch, half-inch thick tablet computer based on the touchscreen interface Apple had debuted three years earlier with its iPhone. An Apple product release is quite unlike anything in the technology world. This is partly due to Jobs' legendary presentation skills. Apple fans and critics alike credit him with emitting a "Reality Distortion Field" that masks shortcomings, makes trivial improvements to Apple products appear revolutionary and leaves all bystanders helpless to resist a lust to acquire new Apple gear. The "RDF" was first attributed to Jobs in the early days of Apple (Hertzfeld 2004), but potency of its effect on the mainstream technology media has reached new heights thanks to a string of Apple successes over the past decade—the iMac, the iPod and the iTunes Music Store, and the iPhone. In the midst of a financial downturn, Apple's business is booming and Jobs was named Fortune's CEO of the Decade.

Jobs describes Apple's approach to building computers as existing at "the intersection of technology and liberal arts." The story goes like this: unlike other computer companies—which are run by geeks with no sense of design and no idea how ordinary people think—Apple takes a more human approach to designing computers. Apple controls the whole experience, designing both the hardware and software to work together in a way that's accessible to non-technical users. Apple products are developed in secret, by a tightknit group of Apple employees in

Cupertino, and unveiled on an unsuspecting world as "magical" and "revolutionary" products from the future.

This vision of Apple is inseparable from Steve Jobs and can be traced back to the founding of the company. Jobs' cofounder, Steve Wozniak, represented the traditional hacker vision of computing. Jobs was the planner, the businessman and the visionary. Wozniak, inventor of the original Apple computer and the Apple II, was the engineer and the hacker. Jobs recognized early on the potential mass market for computers if only a friendly, accessible user interface could be grafted on top of the computer's technical interior. Jobs saw the role of the home computer as an advanced appliance: take it home, turn it on and use it. No knowledge of the inner workings of the machine should be necessary.

Apple is an important case for understanding the nuances of the "moral field" position I am advocating here. In a simple "open"/"closed" taxonomy, the tradition of sealed off, hacker-unfriendly products paint Apple as a "closed" company. However, this is an oversimplified way for thinking about Apple. In an earlier chapter, I highlighted Steven Levy's (2001/1984) distinction between "Hackers" and "Planners" at MIT. Hackers were in it for the joy of hacking, and getting "close to the metal" was an end in itself. Planners, however, were more interested in the social potential of technology and in envisioning ways to transform society with new technology. While Apple has a rocky history with hackers, it is perhaps the quintessential planner company of our era.

Today, mobile platforms are all the rage, and Apple catapulted themselves to the front of the pack with the iPhone. Before the iPhone, smartphones looked like Blackberries—small screens, hardware QWERTY keyboards, email and messaging as the killer apps. After the iPhone, smartphones look, well, like iPhones: large touch screens, minimal hardware buttons, "app stores" with hundreds of thousands of applications to install. Apple has always provoked strong reactions, but since the 2007 introduction of the iPhone, and the iOS operating system that it runs, the reactions have run the gamut. However, both sides of this gamut are defined by a certain kind of populism. Depending on who you listen to, Apple is either the enemy of the people or the peoples' champion. This difference, I argue, can be described, following Levy, as "hacker populism" versus "planner populism." Each populism has a different moral diagnosis for what ails technology, and therefore Apple. Hackers see locked down hardware and proprietary software platforms as a central threat to liberty and creativity in our time. Planners see computers still designed by and for technological elites, infantilizing and intimidating ordinary people and, therefore, limiting their capacity to use technology to improve their lives rather than merely complicating them. Thinking of this in terms of the moral field, this is, in a sense, a battle between the left ("experts") and right ("the public") tails of the "group boundaries" axis. However, as we shall see, much of the action in defining the moral high ground comes from negotiating the other axis: markets and communities.

7.1.1 "Revenge on Developers"

Steve Wozniak, the hacker co-founder, left Apple in 1981 and the first Macintosh, released in 1984, embodied Jobs' vision. The Macintosh was an all-in-one computer and monitor, designed to never be opened by the ordinary user. The Macintosh sported a new pointing device called the mouse and friendly icons to make the computer more accessible to non-technical users. These characteristics endeared Apple to creative artists and writers and made Macs a staple in the education system, and firmly position Apple products within the "computers for everyone" camp. However, longtime hackers saw something more sinister in the Mac's friendly exterior. The very characteristics that Apple claimed made the device easy to use also made it harder to hack. Using unique, custom pieces, from the CPU up to the software, meant that one was limited to Apple-approved components. For a generation raised on the belief that true hacking required access to the low level components of a machine, the Mac was an uninviting black box. Additionally, for all the utopian talk about making computers for everyone, the tightly controlled nature of the Mac had several distinct business advantages for Apple, such as preventing competition from other hardware manufacturers because no one else could build hardware to run the Mac OS.[1]

Over 25 years later, Apple's most recent creations—the iPhone, iPad, and the iOS operating system running on both—bring this early vision into the mobile computing era. The iPhone is an undeniably striking device. Mac blogger John Gruber (2008) gushes:

> If I could travel back 20 years and show my then 15-year-old self just one thing from the future of today, it would be the iPhone. It is our flying cars. Star Trek-style wireless long-distance voice communicator. The content of every major newspaper and magazine in the world. An encyclopedia. Video games. TV. Etc.

[1]There actually was a brief period during the 1990s when Apple licensed it's OS to other hardware companies. The move was almost fatal to Apple and revoking these licenses was one of the first moves Steve Jobs made upon return to Apple in 1997.

The iPhone has also been a huge popular success: over 30 million iPhones have been sold worldwide a little over two years after it's initial launch.

Technologically, the iPhone is an interesting device. Its computational power rivals that of mainstream laptops from just a few years earlier, and the operating system running on the device is a version of Mac OS X, the same exact Unix-based OS running all modern Macs. However, the interaction model for both users and developers is dramatically different for the iPhone. This is true in obvious ways—it's a touchscreen device that fits in your pocket—but also in less obvious ways. What are less obvious changes to non-developers, however, are central to the debates over iOS and Apple by developers, so I'm going to delve into these technical details for a bit.

One of the basic building blocks of modern operating systems is the file. The term "file" itself initially referred to the physical punch cards used by early programmers. In the 1960's virtual "file systems" emerged allowing programmers to store information in multiple "files" on a disk. Shortly after this, *hierarchical* file systems allowed programmers to organize files into "directories", or "folders".

Interacting with files and folders is central to the computing experience on all modern desktop computing platforms. Yet, despite the apparent simplicity of this metaphor, anyone who has ever worked in tech support—or simply served as the resident "computer person" among friends and family—can attest that files and folders are a source of much confusion. The simple question, "Where did you save that file?" can stump a large number of computer users. A common response is to name the application used to edit the file, rather than the location on the file system: "I saved it in Excel." Consequently, many usability experts have long recommended making files and folders invisible to users: let applications handle file management and design a UI around this "I saved it in the application" intuition. iOS takes this advice. As a Unix, of course iOS has a file system, but the file system is inaccessible to end users. Users interact with applications, and the application is responsible for saving and organizing that data on iOS's file system.

Speaking of applications, iOS is a dramatic departure from traditional computers here as well. On regular desktop and laptop computers, anyone can write an application and distribute it themselves, for free if they want. On iOS, the only distribution method for applications is Apple's official App Store. A developer who owns an iPhone and wants to write applications for it—even for purely personal use—must first pay Apple for a one-year, $99 key that allows them to install applications on their own personal iPhone. If they want to distribute this application—for free or for pay—they have to submit the application to the App Store and go through an approval process.

As with the inaccessible file system, the App Store-only policy can be defended on usability grounds. By approving every single app, Apple can guarantee that the iPhone won't fall victim to many of the problems that plague normal PCs, such as viruses that take over your machine for nefarious purposes. Additionally, Apple places limits on what an individual application can do. A single application takes over the screen at a time, limiting the kind of "drag and drop" interoperability between apps made famous by the overlapping window design of the Mac. And while iOS now supports background operations for applications not in the foreground, these operations are a limited subset of that available to a traditional desktop OS.

Most users, if they notice at all, appreciate the simplicity of having all of this handled for them. Expert users and developers, on the other hand, often feel boxed in by these limitations. Even worse, they see Apple's control over the platform not as some benevolent attempt to enhance usability, but as an attempt to lock users and developers into Apple hardware and to prevent competition. For instance, well-known software developer and blogger Dave Weiner says of the iPhone (Weiner 2009):

> The iPhone should have run the same software as the Macintosh. When I first heard about it, I misunderstood and thought I'd be able to write Frontier scripts that ran both on my desktop and the phone. I was a Blackberry user at the time, and I found the idea of a MacPhone truly inspiring. This platform was Apple's revenge on developers. Everything under their control.

In mid-2009, long wait times for App Store approvals and several controversial App Store rejections led to a relatively small, but vocal, backlash among developers and users. For example, Apple has a policy against apps that provide "duplicate functionality" to the built-in Apple products. This means there are no 3rd party email applications available for the iPhone or iPad: you have to use Apple's Mail application that is bundled with the phone. However, Apple bundles several applications with the iOS that are apparently exempt from this policy: countless replacements for Apple's Notes application are available, for instance. Apple's unwillingness to clarify the App Store rules has led many to question Apple's motives.

Plausible technical explanations for this policy exist. Given the constraints on background processing in iOS, Apple may want a monopoly on certain applications to preserve a quality user experience: you want your email application running in the background so it can get new emails as they are sent to you.

Digging a little deeper into Apple's App Store decisions suggests there's more to this story, however. Access to Apple's own iTunes music store is bundled with the device, for example. Amazon, who runs a competing online music download store, has been unable to distribute an Amazon MP3 application for the iPhone. "Duplicate functionality." Google's "Google Voice" application, which allows one to use a separate phone number, provided by Google and independent from the number provided by your iPhone's mobile carrier, to send and recieve phone calls on your cell phone, was rejected by Apple in 2009 then accepted in 2010 after the rejection spurred an FCC investigation into Apple's App Store policies. "In-app purchases" of downloadable content have been available to developers of applications for things like magazines and games—but not music—since summer 2009. Similarly, applications that allow phone-like functionality, such as Skype, have long been available in the App Store. To the conspiratorial eye, Apple's "duplicate functionality" policy appears to refer to Apple's business interests and not technical functionality at all.[2]

In addition to the "duplicate functionality" controversies, Apple has drawn heat for forcing their Puritanical tastes on all App Store applications: beyond a ban on "pornography," this has led to Apple rejecting Dictionary applications that include swear words or ebook applications that allow users to download books Apple deems objectionable. Apple has also banned the use of "Android," a competing mobile platform by Google, in App Store product descriptions. For instance, they rejected an award-winning Android game ported to the iPhone simply for stating that it was an award-winning Android application in the product description.

The take-home point is that iOS is, from a developers point of view, a *closed* platform, even relative to the traditional proprietary OS bogeyman Windows. If you own a Windows PC, you can write and run your own code without having to pay Microsoft for a special key to give you root access to your own hardware. You can distribute this code to anybody according to any terms you choose. And you can use whatever technology you want to develop your application. All of this is true, of course, for Mac OS X as well.

As is always the case with locked down platforms, hackers have managed to break through

[2]I say "conspiratorial eye" because Apple does not make information about App Store rejections public: there may well be solid technical reasons here, but whether or not this is the case is really beside my points here. The more important points are that a) it's always Apple's call, and b) this has bred mistrust of Apple among developers.

the restrictions Apple places on the iPhone. By "jailbreaking" an iPhone, a user can install any applications they choose—including several alternative, unofficial App Stores that have sprung up. For an iPhone user, the process is simple: a one-click affair. Apple has not responded kindly to this practice, deeming it illegal (von Lohman 2009), and even going so far as to suggest jailbreaking poses a terrorist threat (Kravets 2009). iPhone updates frequently break the current jailbreaking techniques, but it usually only takes the jailbreaking community a few days to release an update that works around Apple's latest barriers.

Apple's tight contral of the platform has led to a wave of public criticism by prominent iPhone developers. For example, Joe Hewitt, who had developed the popular Facebook app for the iPhone, announced he would no longer write software for the iPhone (Kincaid 2009):

> My decision to stop iPhone development has had everything to do with Apple's policies. I respect their right to manage their platform however they want, however I am philosophically opposed to the existence of their review process. I am very concerned that they are setting a horrible precedent for other software platforms, and soon gatekeepers will start infesting the lives of every software developer.

This is not a universal reaction, of course. The App Store has been, despite criticism, a massive success. In early 2011, Apple announced over 10 billion apps had been downloaded from the App Store. And this success has ushered in a kind of "Gold Rush" to the App Store, with success stories of out-of-work non-programmers learning how to code on their own and then making a living off iPhone Apps bringing media attention and hype to the platform (Kim 2009).

Additionally, negative reactions the restrictions Apple places on the platform face are tempered by the fact that the device is, ultimately, a phone. It is a communication device and a media consumption device, but its form factor limits the extent to which it can ever be a true platform for content creation and productivity. It's very much a companion device to a computer, not a replacement. This is why Apple's new iPad is, for the purposes of this discussion, even more interesting. Thus far, Apple has applied the same limitations of the iPhone to the iPad—but the iPad is not a phone. It's not a full desktop computer replacement either, but it comes far closer than the iPhone does, and may in fact be a laptop replacment for many users. In other words, the iPad has made clear that Apple sees the iPhone not just as a model for how mobile devices should work, but as a model for how computing should work, and this vision struck a nerve among developers and enthusiasts.

7.1.2 Hacker Populism and Planner Populism

The justifications Apple gave for the restrictions placed upon users and developers by the iPhone—that apps must be Apple-vetted to ensure phone stability and battery life, that apps that "duplicate functionality" cannot be allowed to protect the quality of user experience on a small mobile device, that jailbreaking could make the device dangerous in the hands of terrorists—fall even flatter for the iPad than for the iPhone. The iPad doesn't make phone calls. Prohibiting background processes or alternative email clients on the iPad, but not a MacBook, lacks the plausible technological reasons one can make about a phone. And while the iPhone is a replacement for another phone, an iPad is much more likely to be a replacement for a laptop. With it's larger screen and increased computing power, the iPad can do what all that many, perhaps most, people expect out of a laptop: browse the web, do email, interact with social web services like Facebook and Twitter, watch videos and even create basic word processing documents, spreadsheets or presentations via Apple's iWork applications for the iPad. In other words, if the iPhone could be dismissed as "just a phone", the writing is on the wall with the iPad: this is Apple's vision of the future of computing, not just phones or mp3 players.

This desktop-replacement capability has led to excited commentary that is both critical and celebratory—sometimes by the same individual. The negative and positive reactions are grounded in the two moral models of populism I introduced above, planner populism and hacker populism. Hackers who view themselves as protectors of hacker rights and as members of a community of programmers adopt the "hacker populist" stance towards iOS and the iPad, viewing it with great suspicion. On the other hand, those who view technology as a tool to empower not just programmers, but all people, tend to take the "planner populist" stance towards the iPad, viewing it as a great advance in the democratic potential of computers. Of course, this clean mapping is not a firm line and the exceptions to this rule are what makes the debate interesting, as we shall see.

First, the celebratory, positive reaction and "planner populism." Planners focus on the social impact of computers. They see computers as a liberatory tool for the masses. The technical details of how computers work may be interesting to hackers, but it's what you can do with computers, not how they work, that has revolutionary promise for humanity. Unlike "hacker", "planner" is not a label anyone is going to embrace for themselves. It connotes a top-down, arrogant, know-it-all mentality, so my description of "planner populism" may seem a bit contradictory, but much of the critical edge to "planner" comes from viewing planners from the perspective of hackers and the association of the term "planner" with clueless bureaucracy in American culture. So consider this a caveat to my use of the term here: I do not intend any negative connotiations. "Planner" could just as easily connote a revolutionary, but practical, visionary. And the aims of planners are quite populist: bringing technology to the masses to improve society. Key to this vision is the belief that the revolutionary potential of computers lies in building computers that work the way humans do, that will seamlessly integrate into the people's lives and improve them. In contrast, the hacker ethic argues that understanding the way computers work is key to human empowerment. The act of hacking itself cultivates an ethic that ought to be universal.

The iPad is the descendent of earlier "planner" attempts to create computers with a human face that anyone can use and understand: Doug Englebart's Augment project, the futurists at MIT's AI Lab (if not the MIT Hackers), the PCC and Community Memory, and, of course, the original Macintosh. More than one enthusiastic blogger has even argued the iPad, along with the iPhone, represents the realization of the guide book from Douglas Adams's *Hitchhiker's Guide to the Galaxy*.[3]

The iPad inherits the iPhone's simplified computing experience, which abstracts away all the details about how the underlying computer works. Computer companies have been trying for decades to make computers easier to use, but for the most part they've been very conservative about cutting away the standard desktop OS design elements: files & folders, multiple overlapping windows, a mouse and keyboard, and so on. These elements work together nicely to create a powerful interface for those who have mastered it, but it doesn't scale very well. Microsoft has been building software for "Tablet PCs" for years now that simply substitute a stylus for a mouse. And "netbooks" have became increasingly popular in recent years by taking the standard desktop interface and shrinking it down to a tiny laptop with a tiny screen. All of these attempts are great for niche audiences, but have been usability disasters for mass audiences. They take the same powerful, but complicated, interface paradigm as the standard desktop computer and cram it into a device that makes all the standard OS interactions more difficult (carefully typing on a 3/4 scale keyboard while squinting at tiny

[3]A portable intergalactic encycopedia. The iPad nails two out of three requirements.

fonts of a miniature screen) and do nothing to eliminate the complexity.

By jettisoning this UI baggage in a much more aggressive fashion than others have been willing to do, Apple has created a very intuitive, usable system for people who don't like normal computers. Developers and technologists who focus on software usability and interface design have generally raved about the potential of the iPad. Here are just a few examples:

> Most people don't know what a Web "browser" is, they think the little Internet Explorer icon on their desktop is the "Internet". These people have a computer at home but they don't really need most of it. They struggle with it to just do very basic tasks, like show the latest photos they took to their friends. This is the real mass market, and the iPad is an ideal device for them.
> — Dmitry Fadeyev, Usability Post[4]

> There will be resistance, of that I am sure, I expect many people will feel the way I did years ago towards Macs thinking the iPad is for "dumb people", people who can't handle a real computer. The iPad isn't for dumb people, it is for people who don't want to think about their computer anymore.
> — Caleb Elston, VP Products Justin.tv[5]

> Secretly, I suspect, we technologists quite liked the idea that Normals would be dependent on us for our technological shamanism. Those incantations that only we can perform to heal their computers, those oracular proclamations that we make over the future and the blessings we bestow on purchasing choices.
>
> Ask yourself this: in what other walk of life do grown adults depend on other people to help them buy something? Women often turn to men to help them purchase a car but that's because of the obnoxious misogyny of car dealers, not because ladies worry that the car they buy won't work on their local roads. (Sorry computer/car analogy. My bad.)
>
> I'm often saddened by the infantilising effect of high technology on adults. From being in control of their world, they're thrust back to a childish, mediaeval world in which gremlins appear to torment them and disappear at will and against which magic, spells, and the local witch doctor are their only refuges.

[4]http://www.usabilitypost.com/2010/01/29/on-the-ipad
[5]http://calebelston.com/is-the-ipad-for-dumb-people

With the iPhone OS as incarnated in the iPad, Apple proposes to do something about this, and I mean really do something about it instead of just talking about doing something about it, and the world is going mental.
— Fraser Speirs, educational software consultant[6]

Look at your parents staring at a computer. They can't do double-click. They will never master it. They do not like the mouse. Look at how they never really understood the folder metaphor. They are scared in front of the machine. Clicking with panic. Always at a distance. No love. Just need.

Now give them an iPad. No panic. No fears. They will touch everything. It is so easy. So fast. With my fingers! And when I am wrong, just one click at the one button and it is back home. Safely. A pleasure to use.
— Fabrizio Capobianco, CEO Funambol, a mobile software company[7]

For those of you who think yourselves special because you can set up your own computers (and I am one of them – I could build one from parts if I needed to), get over yourselves. You're not so smart. It has nothing to do with intelligence. It's simply training and experience, even if you're self-taught. You have nothing to hold over those who don't know how to do such things and don't want to.

Personal computing has been stuck far too long in the hobbyist stages. They are complex, error prone, frustratingly buggy devices. Even the best, my beloved Mac, requires a surprising amount of expertise to set up and appear to be easy to use.

Computers are not appliances. But they should be.
— Randy Murray, writer and marketing consultant[8]

When folks need an elevator, we should give them an elevator, not an airplane. We've been giving them airplanes for 30 years, and then laughing at them for being too stupid to fly them right.
— Ed Finkler, web developer[9]

[6]http://speirs.org/blog/2010/1/29/future-shock.html
[7]http://www.funambol.com/blog/capo/2010/01/ipad-scorecard-and-some-thoughts.html
[8]http://whowritesforyou.com/2010/02/02/why-a-computer-is-not-like-a-toaster
[9]http://funkatron.com/posts/were-the-stupid-ones-facebook-google-and-our-failure-as-developers.html

All of these reactions embrace the iPad for its ability to empower ordinary people: a computer that doesn't make them feel stupid or dependent on tech-saavy friends for assistance. From this perspective, the iPad is a populist device, liberating the masses from reliance on smug IT support staff and computers whose complicated interfaces get in the way of what they really need to get done.

On the other side of the aisle, hacker populists see the iPad as representing a dangerous trend towards corporations locking down and controlling the computers that have become integral to our lives. The rich culture and storied traditions of hacking may be cut off from future generations by devices that prohibit hacking. A few select responses from this perspective:

> The thing that bothers me most about the iPad is this: if I had an iPad rather than a real computer as a kid, I'd never be a programmer today. I'd never have had the ability to run whatever stupid, potentially harmful, hugely educational programs I could download or write. I wouldn't have been able to fire up ResEdit and edit out the Mac startup sound so I could tinker on the computer at all hours without waking my parents. The iPad may be a boon to traditional eduction, insofar as it allows for multimedia textbooks and such, but in its current form, it's a detriment to the sort of hacker culture that has propelled the digital economy.
> — Alex Payne, engineer at Twitter[10]

> [My first computer] came with the BASIC programming language pre-installed. You didn't even need to boot a disk operating system. You could turn on the computer and press Ctrl-Reset and you'd get a prompt. And at this prompt, you could type in an entire program, and then type RUN, and it would motherfucking run.
>
> I was 10. That was 27 years ago, but I still remember what it felt like when I realized that you — that I — could get this computer to do anything by typing the right words in the right order and telling it to RUN and it would motherfucking run.
>
> That computer was an Apple][e.
>
> ...

[10]http://al3x.net/2010/01/28/ipad.html

Now, I am aware that you will be able to develop your own programs for the iPad, the same way you can develop for the iPhone today. Anyone can develop! All you need is a Mac, XCode, an iPhone "simulator," and $99 for an auto-expiring developer certificate. The "developer certificate" is really a cryptographic key that (temporarily) allows you (slightly) elevated access to... your own computer. And that's fine — or at least workable — for the developers of today, because they already know that they're developers. But the developers of tomorrow don't know it yet. And without the freedom to tinker, some of them never will.

...

Once upon a time, Apple made the machines that made me who I am. I became who I am by tinkering. Now it seems they're doing everything in their power to stop my kids from finding that sense of wonder. Apple has declared war on the tinkerers of the world.
— Mark Pilgrim, *Tinkerer's Sunset*, programmer and author[11]

Apple's gotten into the habit of acting like you're renting hardware. They've become the all-powerful, over-restrictive, ambivalent IT person in the sky, restricting what users can and can't install on their hardware.

With a device like the iPhone, most people slowly accepted Apple's IT state over time. Apple's stance is basically that their lockdown is for your own good—they're protecting us from unstable apps, pornography, *confusion*, and other nasties.
— Adam Pash, programmer and editor, Lifehacker[12]

The iPhone vision of the mobile Internet's future omits controversy, sex, and freedom, but includes strict limits on who can know what and who can say what. It's a sterile Disney-fied walled garden surrounded by sharp-toothed lawyers. The people who create the apps serve at the landlord's pleasure and fear his anger.

I hate it.

I hate it even though the iPhone hardware and software are great, because freedom's not just another word for anything, nor is it an optional ingredient.

[11] http://diveintomark.org/archives/2010/01/29/tinkerers-sunset
[12] http://lifehacker.com/5458690/the-problem-with-the-apple-ipad

> — Tim Bray, Developer formerly with Sun, announcing a new job with Google with this post.[13]

> What bothers me is that in terms of openness, the iPad is the same as the iPhone, but in terms of form factor, the iPad is essentially a general purpose computer. So it strikes me as a sort of Trojan horse that acculturates users to closed platforms as a viable alternative to open platforms, and not just when it comes to phones (which are closed pretty much across the board). The question we must ask ourselves as computer users is whether the tradeoff in freedom we make to enjoy Apple's superior user experience is worth it.
> — Rafe Colburn, developer and author[14]

Some interesting counterpoints and additions exchanged between the two sides represented here:

Joe Hewitt, the same developer cited above for leaving the iPhone platform due to Apple's App Store policies, on the iPad as in "open" internet device, even if the installed apps are closed, and how the limitations for developers can be good for users:

> The store may not be open, but the iPhone/iPad platform itself could hardly be more open to tinkerers of all ages.
>
> The one thing that makes an iPhone/iPad app "closed" is that it lives in a sandbox, which means it can't just read and write willy-nilly to the file system, access hardware, or interfere with other apps. In my mind, this is one of the best features of the OS. It makes native apps more like web apps, which are similarly sandboxed, and therefore much more secure. On Macs and PCs, you have to re-install the OS every couple years or so just to undo the damage done by apps, but iPhone OS is completely immune to this.
>
> As a developer, it's a bit sad losing the ability to come up with crazy plugins and daemons and system-level utilities, but I believe it's a tradeoff worth making. What people are overlooking is that the Internet is an integral part of the iPhone OS, and it is the part of the OS you can tinker with to your heart's delight. If you want to invent a new scripting language or background service or something,

[13]http://www.tbray.org/ongoing/When/201x/2010/03/15/Joining-Google
[14]http://rc3.org/2010/01/28/is-the-ipad-the-harbinger-of-doom-for-personal-computing/

> you're still totally free to do that, but you're going to have to run it on a web server. If you want total freedom on the client side, then write a web app. You're simply no longer going to be able to tempt users into installing software that corrupts their computer.[15]

Gina Trapani, responding to Mark Pilgrim's post quoted above:

> When you jailbreak your iPhone, is there a *practical* difference between that experience and rooting Android, or flashing your Linksys with open source firmware, or installing XBMC on your Xbox? Is the legality the difference? Which of those activities is legal and which is illegal? (I'm not even sure, which is pretty ignorant on my part, since I've done all of those. I just didn't care.) Is there really a tinkerer's sunset if unlocking the iWhatever in some manner remains an option? If it's not an option, won't the act of tinkering involve MAKING it one? I'm of the mind that if someone wants to tinker, they will tinker, period. Because it's in their DNA, not because it's easy, and because by nature, tinkerers don't play by the rules.[16]

This is an interesting argument because the history of computing is full of hacking open *and* closed systems.

Comedian Steven Fry, who has also became a popular technologist and open source advocate, defended his gushing iPad review:

> Yes, I do like and have tried to champion OpenSource software. How can I square that with my love of Apple? I'm complicated. I'm a human being. I also believe in a mixed economy and mixed nuts. I love our NHS and the National Theatre, but I also love Fortnum and Mason and Hollywood movies.

Each of these populisms is claiming the label of "open". The key question is *open to whom*? Hacker populism demands software and hardware that is open to hackers. If you don't have root access to your system and can't run any code you deem fit to run, the platform is closed. Planner populism wants openness on a higher level: the UI and interaction model for the

[15]http://joehewitt.com/post/ipad/

[16]http://smarterware.org/4941/hackers-dont-tinker-because-they-got-invited

technology needs to be more accessible to non-technical users. Who cares about source code if you have to devote years to learning to program before you can do anything with it? If your iPhone or iPad enables you to do something you couldn't do before, it's more open for your purposes.

The obvious question: are these actually competing definitions of open? Can technology be hacker-friendly on the inside and user-friendly on the outside? Or is there a reason that technology tends to be one or the other? Perhaps Apple products are bold and innovative from a user's perspective precisely because they're built by a small team that can disregard the wishes of a large community of expert contributors who have a more rigid idea of what a computer should look like. That, at least, is part of the argument you hear about why Linux usability suffers. In the late 90s and early 2000s, when Apple was suffering badly, it looked like if any operating system could make in-roads into Microsoft's dominance, it was Linux, the upstart operating system that was driving the internet in data centers around the globe. However, this failed to happen. In the last few years, the Ubuntu Linux distribution has been the highest profile effort at making Linux easy to use, and their efforts are instructive for further contextualizing usability within the moral field.

7.2 Ubuntu

Apple is content to alienate the "hacker populists" if it means satisfying their "planner populist" impulses. However, for open source projects seeking mainstream success, making such a clear choice is not possible. Several Linux distributions have attempted to make mass market inroads with mostly lackluster results. The most notable effort in recent years has come from Ubuntu, founded in 2004 by South African entrepreneur Mark Shuttleworth with the explicit mission of making Linux more easy to use. However, putting this mission into practice often entails making decisions that run contrary to either the community ethos of open source, the traditional expert user-base of Linux distributions, or both. The result is a situation where even the smallest decisions about the distribution stir up moral deliberation over the nature of the relationship between the community, the market, and those in authority positions for the distribution.

In this section, I will discuss what, to an outsider, would appear to be purely technical decisions—which software to include in Ubuntu, and the design of Ubuntu's user interface—but are, in fact, hotly contested within the community because of the way they engage the fundamental tensions within the moral field.

7.2.1 Usability, Freedom, and File Formats

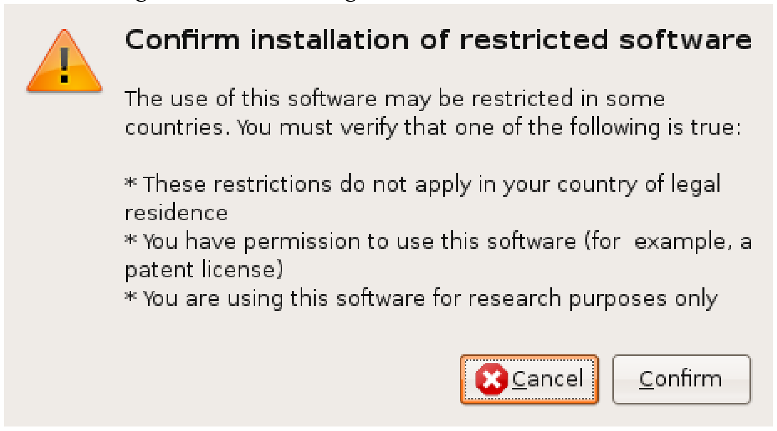
From a moral perspective, free software and proprietary software seem like oil and water, two things that just cannot mix. Technologically, though, this is not the case. Moral and legal boundaries notwithstanding, one can write closed source software to run on Linux, and in fact, to make a Linux distribution do the things that most people expect from a computer these days requires some proprietary software. For instance, the popular mp3 file format is not an "open format": anyone distributing the encoders or decoders necessary to create or play mp3 files is legally obligated to pay license fees to the patent holder, the Fraunhofer Institute. A similar situation exists for the H.264 standard behind much of the video found on the internet. The codecs to decode these formats are available for Linux, but because software patents run contrary to the free and open source ethic, most distributions choose to not pay the license fees and ship their distributions without these codecs. Instead, they install codecs for free, but far less widely used, formats by default, such as the Ogg Vorbis and Theora audio and video formats.

But playing music and video files is a key part of what most people expect a modern computer to do. The result is a classic dilemma for social movements: compromise on the purity of software in your distribution to help win the larger battle for greater adoption of open source operating systems, or stick to your principles and only include free software in the distribution despite the usability costs? For example, Linspire, a Linux distribution that included proprietary software in an attempt to succeed in the consumer market, prompted Richard Stallman to assert:

> No other GNU/Linux distribution has backslided so far away from freedom. Switching from MS Windows to Linspire does not bring you to freedom, it just gets you a different master (Matzan 2005).

Ubuntu does not include codecs for these media files by default, but makes the installation of these codecs a one-click operation after a stern warning (see Figure 7.1).

Figure 7.1: Installing Media Codecs in Ubuntu



Confirm installation of restricted software

The use of this software may be restricted in some countries. You must verify that one of the following is true:

* These restrictions do not apply in your country of legal residence
* You have permission to use this software (for example, a patent license)
* You are using this software for research purposes only

A similar situation exists for other formats such as Adobe's PDF file format and Microsoft Office's ubiquitous .doc, .xls, and .ppt formats. Most computer users have their life's work saved in one of these formats and cannot seriously consider any computer lacking full support for these formats. In the case of PDF, Adobe does make a version of their Reader software, but it is closed source and therefore not installed by default in most distributions. Likewise with Adobe's Flash player. A 2010 usability study by Canonical found that these decisions about pre-installed software—made for principled, ethical reasons that resonate within the free software community—have a usability cost with non-expert users:

> Most users of Ubuntu will need at one point or another to get flash in order to view a website. When attempting to download a flash player, participants had to choose between, YUM, .tar, .gy, .rpm, .deb or APT. Most didn't know what to do at that point. No-one succeeded in downloading the flash player (Poirier 2010).

Microsoft does not make a version of Office for Linux, but the free OpenOffice suite will read files created by Office, but often not perfectly: Office uses Microsoft's proprietary formats and free, complete documentation doesn't exist. Additionally, while OpenOffice can read and write Microsoft .doc files, its default file format is the Open Document Format (ODF), a free and open file format standard. While supporting Word files is a pragmatic step towards interoperability, the ideal open source community solution is to push people towards using ODF. Canonical's usability testing found that this can be off-putting to new Linux users:

> Almost all our participants habitually used Microsoft Office. As they explored OpenOffice, one of their main concerns was the degree of its compatibility with other software. One participant asked: “How compatible is ODF?”, while another wondered: “When someone else receives my document will it look the same [as when I sent it]?” (Poirier 2010)

Drivers for graphics cards are another problem as the dominant graphics cards vendors—ATI and NVIDIA—only release either closed source drivers or inferior open source drivers, so users wishing to install these drivers see a dialog box similar to the warning about media codecs in Figure 7.1.

7.2.2 Usable Design

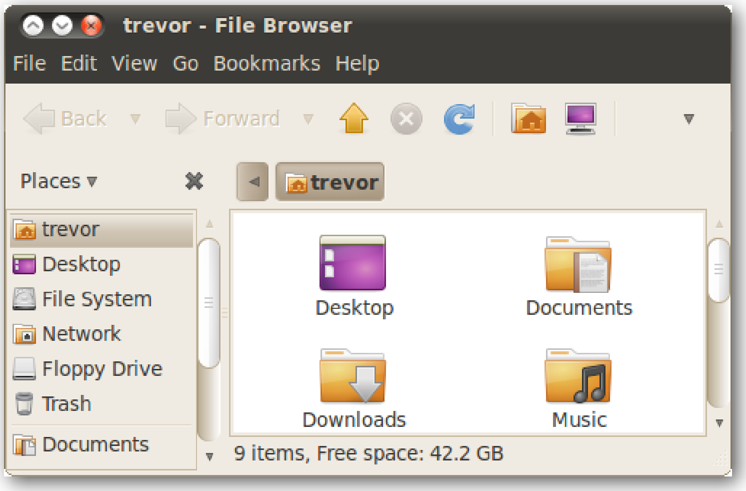
Making Linux more usable is about more than what software to include: it’s about giving that software a user interface that people find appealing and intuitive. The commonly-heard knock on open source software is that open source is great for producing infrastructure components like operating system kernels, programming languages and web servers, but the decentralized, voluntary, and often chaotic process that works for these kind of open source projects simply cannot produce a beautiful, well-designed user interface (UI) that appeals to non-technical users. In recent years, the resurgence of Apple after Steve Jobs’ return to the company has made Apple the exemplar for good UI design: a closed, tight-knit group inside Apple secretly develops and then unveils beautiful, polished, closed source products that receive rave reviews for their usability. In contrast, open source projects develop their interface in the open and often lack the guidance of usability experts to ensure clarity and consistency, let alone beauty, in the resulting products. In an oft-cited essay on the topic, Matthew Paul Thomas called this the “too many cooks” problem:

> In the absence of dedicated designers, many contributors to a project try to contribute to human interface design, regardless of how much they know about the subject. And multiple designers leads to inconsistency, both in vision and in detail. The quality of an interface design is inversely proportional to the number of designers. (Thomas 2008)

Assuming that this argument against open source design is true, we again see that making Linux “user-friendly” involves not just technical decisions, but taking stands on moral dilemmas inside the field. To improve the usability of Ubuntu, Mark Shuttleworth launched Project Ayatana and the Canonical Design Team by hiring a dedicated team of designers and usability experts, including Matthew Paul Thomas. While it would be an exaggeration to say Canonical's efforts are emulating entirely the secretive, closed development model of Apple—for example, the relatively innocuous act of starting an invitation-only mailing list for the project lead to criticism from the community—these are efforts to centralize design decisions in order to create a more visually pleasing, consistent and clear interface tying the disparate elements of a Linux distribution together.

The results of the design team's work are beginning to make an appearance. Ubuntu's 10.04 release in Spring 2010 included a brand new visual theme for the operating system. One dramatic change for the new look was the movement of the window control buttons (for close, minimize, and maximize) from the familiar top-right corner (their placement in Windows) to the top-left corner in a rearranged order: maximize, minimize, then close on the inside (see Figure 7.2).

Figure 7.2: Window Buttons in Ubuntu 10.04

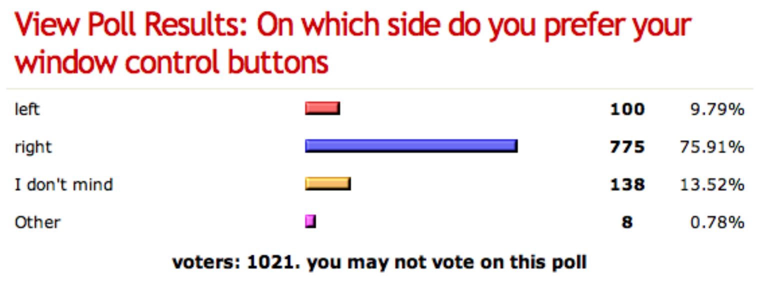


The change prompted an immediate outcry from Ubuntu developers and users, upset that a change to such a longstanding UI element could be unilaterally forced on them—"At the very least Ubuntu should allow this to be optional. These types of lock-in changes are the hallmark of Apple, not an open source community".[17] The Ubuntu Forums were filled with discussion

[17]http://ftbeowulf.wordpress.com/2010/03/21/ubuntu-10-04-window-controls/

threads protesting the move and documenting how to move the buttons back. Forum polls were conducted to express displeasure (Figure 7.3).

Figure 7.3: Ubuntu Forums Window Button Poll



View Poll Results: On which side do you prefer your window control buttons

left	100	9.79%
right	775	75.91%
I don't mind	138	13.52%
Other	8	0.78%

voters: 1021. you may not vote on this poll

This change was submitted as a bug to Ubuntu's bug tracker, with the request that the buttons be moved back to their original position. Shuttleworth responded:

> This is not a democracy. Good feedback, good data, are welcome. But we are not voting on design decisions (Shuttleworth 2010).

Shuttleworth elaborates:

> This is a difference between Ubuntu and several other community distributions. It may feel less democratic, but it's more meritocratic, and most importantly it means (a) we should have the best people making any given decision, and (b) it's worth investing your time to become the best person to make certain decisions, because you should have that competence recognised and rewarded with the freedom to make hard decisions and not get second-guessed all the time.

Shuttleworth's comments evoked quite a reaction. Many were supportive and defended Shuttleworth:

> I trust a qualified doctor to prescribe my medicine based on what he knows. I don't then expect a poll of the waiting room patients to see whether they agree with him; My trust is in people with interface design backgrounds to choose the best interface and design options.[18]

[18]http://www.omgubuntu.co.uk/2010/03/why-mark-shuttleworth-is-right-ubuntu-is-not-a-democracy-and-nor-should-it-be/

The "emulate Apple" strategy was no secret either:

> The trouble with democracy in the open-source world is that the result is often muddled and mediocre. That "happy medium" isn't all too happy after all; the companies with the gutsiest (and often, least democratic) decision-making processes are those which often take the lead. And if you're thinking Apple, you're not the only one.[19]

However, there was much disagreement as well, particularly Shuttleworth's characterization of Ubuntu as "not a democracy." For example, here are a few comments from a thread on LXer, a Linux news discussion forum[20]:

> Mr. Shuttleworth's "Jobsian" "you get to comment, but we decide" certainly doesn't cut it. If it did, I wouldn't have left Microsoft in the first place or I'd have switched to Apple.

Another recalled the debate over proprietary drivers discussed above:

> What originally turned me off to Ubuntu (the first distro that I actually worked as a replacement for Windows) was the debate about whether proprietary video drivers would be installed by default. We had the same "this is not a democracy" thing at that time. I really learned to appreciate the importance of choice at that time. I had a video card that didn't work properly with the proprietary driver, so Ubuntu would essentially have been useless. Yet Mark S. made it clear that he would do what he wanted to do. Shortly after that, I started playing with Debian, and it was good.

On the tension between authority figures making important decisions and then soliciting free assistance from the community for testing:

> Interesting. I just went to the Ubuntu website. It says "If you want to help us test it, download beta 1 now. We appreciate your feedback and support"

[19]http://downloadsquad.switched.com/2010/03/19/mark-shuttleworth-clarifies-ubuntu-is-not-a-democracy/
[20]http://lxer.com/module/forums/t/30456/

> This is an appropriate time for an extension of the middle finger. If Mark is so smart, he can do it himself. Why would I waste my time improving his for profit OS? He's like a dog biting the hand that feeds him.

Another commenter identified one of my central claims about the moral field: it might take alienation of the community to reach a wider audience:

> Ubuntu's No. 1 problem is that if it builds for the fanboy, he is happy; but if it doesn't build for the Windows/Mac user, they'll never even consider making the switch. Can one system satisfy both?

And as the language here demonstrates: the "fanboy" wil take this as a betrayal of community values.

In the particular case of the window buttons, some concessions were eventually made: by the final release, the button order reverted to the tradition "close, minimize, maximize" order.[21] The incident reveals how a simple UI change can require project leaders to negotiate their position on the moral field. Exercising too much top-down decision-making can give the perception of abandoning "the community" and taking on the characteristics of commercial, closed design teams like those found at companies like Apple. And making modifications to an interface that current, more tech-savvy users like and are accustomed to can lead to alienating the expert base of the distribution as they strive to create something appealing to a broader public audience.

And the window buttons were just the start. One year later, in Spring 2011, Ubuntu 11.04 brought the release of "Unity", Canonical's replacement for the standard GNOME desktop that Ubuntu had ran by default. Unity is a strong departure from the traditional GNOME desktop, adopting a Mac OS X-style menubar and application Dock, and a more minimalist feel overall designed to be more accessible to non-Linux users. Here's how Mark Shuttleworth described their approach with Unity:

> We put user's first because we committed to test and iterate Unity's design with real users, and evolve it based on those findings. We've documented the

[21]However, the new button *position*, now on the left side of the window, did not change. It was later revealed that this move facilitated a new notification system to appear on the right side of the window in future versions of Ubuntu.

> process we're following in that regard, so that other free software projects can decide for themselves if they also want to bring professional design into their process. I very much hope that this will become standard practice across all of free software, because in my view the future of free software is no longer just about inner beauty (architecture, performance, efficiency) it's also about usability and style.[22]

However, as Bruce Byfield put it, Unity was a "tragically ironic" choice for a name: the adoption of Unity led to a backlash among many in the open source community who believed Ubuntu was "doing their own thing" instead of being a good community participant.[23] On one side, GNOME developers argued Canonical abandoned the open source community that it leaned on while building a user base during it's early years. On the other hand, defenders of Canonical pointed to the disadvantages of an open source community when it comes to design:

> Unity is progressing nicely without the bureaucratic burden of the Gnome hierarchy.... With Canonical they can hire UI experts and carry out tests to make important decisions, than relying on a developer who is out of touch with regular users (who aren't programmers) of the application.[24]

In short, interface decisions or choices about what software to include in the distribution force Ubuntu to engage with both dimensions of the moral field. Which group boundaries are they observing? Are they creating a *product* to compete in the *market*, or are they leading a *project* building a *community resource*? The Ubuntu Design Team cannot actually answer these questions one way or the other. They are doing all of these things, and "open source" enables a hedge on these hard questions. Closed decision making or including proprietary software can be viewed as enabling a broader public reach of software freedom or as a betrayal of free software values; an example of how open source can succeed in the market or as the market encroaching upon the sacred values of free software. My point is not that Canonical has to choose one vision over the other, but that the project of building software involves engaging with groups and individuals across these contentious discourses.

[22]http://www.markshuttleworth.com/archives/671

[23]http://itmanagement.earthweb.com/osrc/article.php/3920156/Ubuntus-Unity-Desktop-Tragically-Ironic-Product-Name.htm

[24]http://blogs.gnome.org/bolsh/2011/03/07/has-gnome-rejected-canonical-help/#comment-3475

7.2.3 Funding Usability

When Shuttleworth launched Ubuntu and Canonical in 2004, he knew it wouldn't be a profitable business for some time, announcing a five year plan to reach profitability. Seven years later, Ubuntu is the most widely used Linux distribution by an impressive margin (it has occupied the #1 spot on distribution-tracker distrowatch.com for several years now) but Shuttleworth is now saying profitability may still be a couple of years off.[25] How do you make money producing something as complex as an operating system and giving it away for free? The answer that produced the first wave of Linux distro-based business—most notably Red Hat—is "support". One of the reasons companies are willing to pay large fees to Microsoft for software is that when something breaks, they have a service contract and can get assistance. Red Hat fills this role for Linux in businesses, as does Canonical, who sells support for their server and desktop versions of Ubuntu. However, in recent years, Canonical has introduced consumer-level features into Ubuntu in an effort to make money. For instance, Ubuntu includes an integrated cloud service called Ubuntu One that allows Ubuntu users, for a monthly fee, to sync files between computers and store backups in the cloud. Additionally, an Ubuntu One Music Store that enables MP3 purchases from within the default Ubuntu music application. These features, it can also be argued, are about usability as well. Internet file storage and music purchases are standard features on proprietary competitors such as Mac OS X, and they each improve the experience for end users of the OS. But, for our purposes here, the relevance to usability to somewhat tangential: these are usability improvements in the service of providing Canonical with a business model.

However, this Ubuntu One Music Store, in particular, has lead to some controversy within the open source community. In version 11.04, Ubuntu switched their default audio player from Rhythmbox to Banshee. However, Banshee already had a built-in music store which hooked into the Amazon MP3 store. Banshee donated the affiliate revenue from music purchases in this store to GNOME, the non-profit organization that produces the desktop environment used by Ubuntu and many other Linux distributions. Canonical, however, replaced Banshee's store with their own Ubuntu One store, whose proceeds go solely to Canonical. In response to criticism from Banshee developers—who were unaware of the change until beta versions of Ubuntu revealed the change—as well as prominent open source figures such as Larry Ewing[26]

[25]http://www.devside.net/articles/ubuntu-linux-dying
[26]http://twitter.com/lewing/status/40871058521325568

and GNOME founder Miguel De Icaza[27], Canonical announced they would donate 25% of the revenue to GNOME, keeping 75% for Canonical. This rather tone-deaf response, if anything, lead to even more criticism.

This case points to the core tension in trying to commercialize open source software: the mix between volunteer effort and business interests makes for a volatile mix. Banshee, for example, was produced with the support of Novell, also a large, for-profit technology company. However, Novell decided not to commercialize the selling of music within Banshee, probably because they did not need to: Banshee's developers revealed that the affiliate revenue from the store raised about $15,000 per year for GNOME.[28] For a large company like Novell, this is not an enormous amount of money. For a non-profit like GNOME, or a relatively small company like Canonical, however, it's apparently worth fighting over.

It also reveals a long-standing community criticism of Canonical: that "Canonical doesn't contribute." While companies such as Red Hat, IBM, and Novell have long funded many of the core contributors to the Linux kernel, for instance, Canonical has not.[29] Here's how Shuttleworth defends Ubuntu's approach:

> We focus most of our effort on integration. Our competitors turn that into "Canonical doesn't contribute" but it's more accurate to say we measure our contribution in the effectiveness with which we get the latest stable work of upstream, with security maintenance, to the widest possible audience for testing and love. To my mind, that's a huge contribution.[30]

In other words, Ubuntu is an effort to pull various Linux packages together into a more usable form. However, this opens Canonical up to criticisms about exploiting "upstream" projects when they attempt to commercialize what they do. For instance, Debian developer Lucas Nussbaum compared Ubuntu's Banshee package to the Debian Banshee package it was derived from:[31]

> For those wondering how much work was done by Canonical directly on the Banshee package: the banshee package in Ubuntu natty is based on the package

[27]http://twitter.com/migueldeicaza/status/40878260325859328
[28]http://banshee.fm/about/revenue/
[29]http://www.linuxfoundation.org/publications/whowriteslinux.pdf
[30]http://www.markshuttleworth.com/archives/162
[31]http://www.lucas-nussbaum.net/blog/?p=656

> currently in Debian experimental. The package is mainly maintained in Debian by an Ubuntu developer not paid by Canonical AFAIK, Chow Loong Jin. There are some differences between the Debian package and the Ubuntu package, which are fairly limited (diff here) and mainly about enabling UbuntuOne and disabling the other music stores. That patch itself apparently was provided by Jo Shields, who doesn't seem to be a canonical employee. (Feel free to correct me)
>
> I think that one of the conclusions to draw from this story is that we now have several proofs that Ubuntu isn't a volunteer-driven project, and that volunteer contributors should really decide whether they are OK with working for free for Canonical, or if their free time would be better spent on other projects where they actually have a chance to influence decisions.

Canonical's practice of taking a cut of affiliate revenue in open source software bundled with Ubuntu is not limited to Banshee either:[32]

> The company also swaps in its Amazon affiliate ID for the Amazon search in Firefox.
>
> Does Canonical deserve a cut of the revenue for distributing Banshee and Firefox? There's an argument to be made that Banshee and Firefox wouldn't be in front of those users in the first place if it weren't for Canonical. But ask yourself whether this would seem Kosher if Microsoft or Apple toggled the bits through an update and replaced the Firefox affiliate code when running on their platforms. I'm guessing that scheme would have very few supporters.

The point here is that whether or not Ubuntu is villified or praised seems to largely depend on the answer to a few questions: a) what counts as a valid contribution to the open source community?, and b) when is it justified to profit off of others' code?

As with the issue of licensing, we have seen that decisions about software design and usability are not just technical decisions. They require navigating a complicated terrain of beliefs about community values, the market, and fairness. These deliberations are so heated because these are moral issues. The decisions reached are not just interpreted as beneficial or detrimental to the

[32]http://www.networkworld.com/community/node/71460

fate of the project or product, but as good or bad, right or wrong, moral or immoral. However, these moral readings are not just the result of abstract reasoning over moral principles. As the classic Miles' Law (Miles 1978) puts it, "where you stand depends on where you sit," but "where you sit" in the diffuse, overlapping world of businesses, hobbyists, non-profits, activists, and end users that make up "the open source community" is a complicated question.

Chapter 8

Conclusion

Chapter Preview: In this chapter, I will briefly review the theoretical and empirical contributions this dissertation makes to sociology as a discipline as well as to a general social scientific understanding of free and open source software development. I will conclude by arguing that this dissertation provides a foundation upon much future research can be conducted, and by suggesting the shape some of that research could take.

In this dissertation, I have constructed a theoretical framework, grounded in current theory and research in sociology, for making sense of the vigorous, emotion-laden debates going on in software development about "openness" and "freedom." This framework emerged out of a conceptual narrative (Somers 1994) that shows the development of the moral field, from the origin of the MIT hackers through the homebrew hackers of the 70s, the ascendance of the PC industry in the 1980s, and the institutionalization of free and open source software in the 90s.

Next, I explored two contemporary areas of debate in computing—licensing and usability—drawing on four case studies. First, I showed that the issue of software licensing is not merely about pragmatic, legal matters, but is a deeply moral issue, shaped by differing views about human motivations and obligations, as well as competing theories of how well the market can be trusted to ensure fairness and efficiency. I looked at the Android mobile operating system and argued that the licensing decisions made by Google were shaped by their needs in the particular market they were entering—the mobile handset market. I showed how their rhetoric around this licensing played up the moralization of licensing by painting Google as a champion of openness, but set the stage for cries of foul play when Google sided with a definition of openness in their own market interest rather than the moral interests of the free software community. I then turned to the WordPress content management system. WordPress is significant because it reveals how technological changes—the move towards web applications and "the cloud"—pose challenges for advocates of free software by enabling business models (such as "Software for a Service") that some argue are antithetical to the moral obligations of the GPL.

I also examined debates over usability and open source software. The open source model, conventional wisdom says, works best for infrastructure projects (programming languages, system tools) but falls victim to the "too many cooks" problem when it comes to designing elegant, easy-to-use interfaces. Companies that adopt a closed development model with respect to usability—I looked at Apple as the exemplar of this approach—are criticized for threatening the open source culture hackers love and thrive in, yet defenders of this approach (who I dub "Planner Populists" in opposition to the "Hacker Populists") see a different definition of openness here: freedom for end users who want to *do* something with technology beyond just building it themselves. Following Apple's lead, Canonical has aggressively sought to improve the usability of their Ubuntu Linux distribution by assembling a relative small, closed team of designers and user experience experts to help them humanize the user experience in Linux. Their efforts, however, prove controversial—*morally* controversial. Their efforts at changing the user interface draws criticism from a Linux community that uses Linux specifically because it's an operating system built by and for them: experts who want to tweak everything about their system. Therefore, in their eyes, Ubuntu is betraying the community's wishes, and, worse, exploiting the decades of hard work spent building Linux by pushing its development in a more commercial, more closed, direction.

In short, this dissertation has shown how the technology that has worked its way into our everyday lives is developed in a social environment with a vibrant and contentious moral culture. It is a culture that views the means of production and distribution of these technological objects as a pressing moral concern. In the information age, hackers theorize, control over the tools of information is vital for people to benefit from technology, rather than be controlled by it. Variants of this belief, however, are used to justify a wide range of positions: for pushing open source software into the market, and also for protecting open source code from the market; for making technology easier to use for non-technical users, as well as for keeping technology complicated and messy but maximally open to end user improvement and self-learning. These positions are not ultimately just about software, however. They're about *people*: about competing beliefs about what is good for communities, markets, and society. Software is just the medium here for the to work through these "metanarratives"—an increasingly important medium. Understanding where this powerful force comes from is crucial to understanding the potential, and potential blind spots, of those building this disruptive technology.

Theoretically, my approach here has straddled two approaches. On the one hand, research on group boundaries that views *boundary work* as central to understanding social life, and on the other hand, various bodies of research in cultural sociology on political discourse and on markets. In my historical and contemporary chapters, I have brought these two approaches together by illustrating how discourse about democracy, and democratic concepts such as "openness" and "freedom," and ideas about how markets and communities ought to work, function as a medium through which different groups draw boundaries around the kind of code they're writing and why it's the *right* or *wrong* way—in both a practical, but also moral sense—to develop software. Members in a voluntary community of expert hackers may approach these questions in a different way than managers at an IT company seeking to use open source in their proprietary services or products. Yet liberalism's contradictory embrace and suspicion of both strong markets as well as strong communities, provides a glue for discussing these issues. In the case of software, this means talking about things like "openness" and "freedom."

I have used the concept of "field" to describe the way that two dimensions of social life—group boundaries and cultural discourses about community and markets—structure the way that individuals and groups within open source communities, and the larger computing field as well, define what they are doing and position themselves relative to others. We saw several debates within computing about "open" vs. "closed" development, but these terms are vaguely defined. For instance, we saw opponents of Apple criticizing the "closed" nature of Apple's devices and opponents of Ubuntu's work on the user interface for Ubuntu as being excessively "closed." On the other hand, some praised Android's "openness" while others criticized the ways in which Android is "closed" after all. This "open"/"closed" distinction, however, is too thin to capture what is really going on in these debates. A closer look reveals that people's disagreements about what is "open" or "closed" amounts to disagreements about where to draw group boundaries and their moral vision about community and the market. Crucially, these are not just two unrelated sources of disagreement, but these dimensions are connected by forming four distinct models about what software development ought to be all about. Companies just as Microsoft and Apple embody the "market/public" orientation towards computing, while the Free Software Foundation and the early days of the Linux kernel represent the ideal "community/expert" orientation towards computing. Much of the action in open source these days, however, shows projects attempting to straddle quadrants of the moral field: Ubuntu attempts to remain loyal to the open source community while pushing the distribution in a more mass market direction. Android claims the mantle of the first mass market open source operating system while at the same time drawing sharp criticism for their betrayal of open source ideals, and open source practice, in the process. In these debates, "open" is used a proxy to criticize betrayal of community values and loyalty. Additionally, we saw the clear cases aren't so simple either: from the perspective of building usable machines that empower ordinary people to contral their lives, Apple's closed development process may, in fact, produce more "open" products for a large body of users.

One of the strengths of the concept of field is that it scales. For the most part this paper has zoomed in on the culture of computing at a particular level: the level at which open source projects intersect with wider field of computing. But an obvious question here is: just computing? Am I simply presenting a framework for making sense of what's happening in this single domain or am I using computing to develop a broader theory of social behiavor applicable to other communities and domains of activity. Is a *moral field* a good way to understand other spheres of action as well? I am leaving this as an open question. The moral field framework does have some theoretical features however that may make it portable to other domains. In particular, social theorists have long been preoccupied with reconciling structural and cultural factors. I am borrowing the language of "field" from Pierre Bourdieu who developed the concept specifically to find a way of making sense of how structure and culture interact to shape social phenomenon. So my work here is another effort in this long theoretical tradition, and zooming out to view how this particular computing field interacts with other fields—the legal field, the political field, the media field—and exploring whether or not the particular constellation of dimensions I have described for computing are represent these intersecting fields as well is a project for future work.

This dissertation does not pretend to be the definitive statement on the moral and political culture of open source and computing. Rather, I am intending this to be a launching pad for further thinking and research about this subject. As I reviewed in Part I, many have noticed the striking culture software developers have spawned. My contribution has been to place this culture in a larger framework that helps us understand computing culture in the context of other political and moral cultures, and also helps guide future research on the topic. For instance:

- Analysis of project bug trackers to understand, quantitatively and qualitatively, how the criteria for what counts as a "bug" is informed by the moral visions and group boundaries represented by the moral field.

- In-depth analysis of open source stewardship by for-profit corporations. For instance, in 2010, Oracle acquired Sun Microsystems, and with it a large number of open source projects Sun had sponsored. Oracle is not as open source-friendly as Sun, however, and the fate of these open source projects, which is still unfolding, will make a fascinating case study. For example, OpenOffice, a Microsoft Office alternative developed by Sun, has forked into LibreOffice out of opposition to Oracle's policies. MySQL, a hugely successful open source database acquired first by Sun and now Oracle, has been forked by its co-founder, Monty Widenius, out of fear of what Oracle will do with MySQL.

- In-depth interviews. In chapter 3, I discussed how I initially considered in-depth interviews but decided to start by analyzing the large amount of existing discourse available online. Now, however, in-depth interviews would be a fruitful way to follow-up on my findings here. In-depth interviews about specific events like those covered here—Ubuntu's Unity project, Google's Android, or many other similar cases—could uncover deeper nuances about how people think about these cases and test, or disprove, my assertion that the moral field not only describes the social-level discourse we see around open source, but also shapes individual self-definitions of the field. Additionally, this could address a bias in focusing on existing discourse: the risk of overvaluing those who talk a lot. It may be the case that those who love to participate in debates over the political and moral aspects of code are fundamentally different than a possible "silent majority" that feels differently, or indifferently, about these issues.

- Internationalization is a huge force in open source over the last decade. While the famous projects of the 90s and early 2000s were dominated by Americans and Europeans, open source communities in Latin America, Asia, and Africa are currently thriving. If open source culture is tied to modern liberalism in Western nations, how does this change in the context of China, Brazil, or Kenya?

- Gender and open source culture. I know from attending conferences and reading online that there are many unpublished studies of gender dynamics in open source on the way in the next few years. As I mentioned earlier, studies indicate that women are less represented in open source projects than in proprietary software. Several open source groups, such as Debian Women, are organizing to combat this inequality and gender in open source, and "geek culture" more generally, is going to be a vibrant field of research in the coming years.

For each of these, the moral field framework developed here suggests a focus on the relationship between moral visions and group boundaries. Moral visions of communities and markets, and group boundaries and boundary processes defining who belongs to entities like the community or the public, and what actions are in the best interest of these groups. It's easy to slide into treating one of these factors as dominant, viewing either all conflict as a "battle of ideas" or as irreconcilable differences between interest groups. Neither is reducible to the other, however. The wiggle room in interpretation of the ideas—"openness," "freedom"—enables and constrains communication across and about group boundaries, and vice versa. Appeals to the requirements of the market or to the necessities of programming efficiency necessarily entail claims about the moral virtues of an open source commons or integrating code into the free market. The interplay between notions such as democracy and the market, freedom and work, personal skill and enlightened cooperation, and the real human relations that collaborate on building software, big and small, constitutes a field of interaction that is complex, dynamic, and a rich testbed for the core moral dramas of our time.

Bibliography

Abend, Gabriel. 2010. "What's New and What's Old about the New Sociology of Morality." In *Handbook of the Sociology of Morality*, edited by Steven Hitlin and Stephen Vaisey, pp. 561–584. New York: Springer.

Asay, Matt. 2009. "Apache better than GPL for open-source business?" *The Open Road*. Retrieved October 5, 2011. (`http://news.cnet.com/8301-13505_3-10229817-16.html`).

Bail, Christopher A. 2008. "The Configuration of Symbolic Boundaries against Immigrants in Europe." *American Sociological Review* 73:pp. 37–59. ISSN 00031224. (`http://www.jstor.org/stable/25472513`).

Barth, Frederick. 1969. "Introduction." In *The Social Organization of Culture Difference*, edited by Frederick Barth, chapter 1, pp. 9–38. Little, Brown and Co.

Becker, Howard S. 1995. "Moral Entrepreneurs: The Creation and Enforcement of Deviant Categories." In *Deviance: A Symbolic Interactionist Approach*, edited by Nancy J. Herman, p. 169. Lanham, MD: Rowman and Littlefield.

Benkler, Yochai. 2006. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press. ISBN 0300110561.

Berry, David M. 2008. *Copy, Rip, Burn: The Politics of Copyleft and Open Source*. London: Pluto Press.

Blasi, Augusto. 1984. "Moral Identity: Its Role in Moral Functioning." In *Morality, Moral Behavior, and Moral Development*, edited by W.M. Kurtines and J.L. Gewirtz, pp. 128–139. New York: Wiley.

Bourdieu, Pierre. 1983. “The field of cultural production, or: The economic world reversed.” *Poetics* 12:311–356.

———. 1984. *Distinction: A Social Critique of the Judgment of Taste*. Cambridge: Harvard University Press.

———. 1992. *Logic of Practice*. Stanford University Press.

boyd, danah and Eszter Hargittai. 2010. “Facebook privacy settings: Who cares?” *First Monday* 15.

Breindl, Yana. 2011. “Promoting Openness by “Patching” European Directives: A Five-Stage Adoption Process.” *Journal of Information Technology and Politics* 8.

Bridges, Tristan S. 2009. “Gender Capital and Male Bodybuilders.” *Body & Society* 15:83–107. doi:10.1177/1357034X08100148.

Brubaker, Rogers. 2004. *Ethnicity Without Groups*. Cambridge, MA: Harvard University Press.

Buskirk, Eliot Van. 2009. “LimeWire Creator Brings Open-Source Approach to Urban Planning.” *Wired.com*. Retrieved January 4, 2010. (`http://blog.wired.com/business/2009/01/mark-gorton-ceo.html`).

Castells, Manuel. 1996. *Rise of The Network Society (Castells, Manuel. Information Age, 1.) (Vol 1)*. Wiley. ISBN 1557866171.

———. 2001. *The Internet Galaxy: Reflections on the Internet, Business, and Society*. Oxford University Press. ISBN 0199241538.

Cheney, George. 2002. *Values at Work: Employee Participation Meets Market Pressure at Mondragon*. Ithaca, NY: Cornell University Press.

Coleman, E. Gabriella and Alex Golub. 2008. “Hacker practice: Moral genres and the cultural articulation of liberalism.” *Anthropological Theory* 8:255–277.

Coleman, Gabriella. 2004. “The Political Agnosticism of Free and Open Source Software and the Inadvertent Politics of Contrast.” *Anthropological Quarterly* 77:507–519.

———. 2010. “The Hacker Conference: A Ritual Condensation and Celebration of a Lifeworld.” *Anthropological Quarterly* 83:47–72.

Coleman, Gabriella and Mako Hill. 2004. “How Free Became Open and Everything Else Under the Sun.” *M/C Journal* 7. (`http://journal.media-culture.org.au/0406/02_Coleman-Hill.php`).

Congress, U.S. 2005. “Using Open-Source Information Effectively.” *Committee on Homeland Security*. Retrieved January 4, 2010. (`http://purl.access.gpo.gov/GPO/LPS81050`).

Cook, Ben. 2010a. “GPL in Practice: an Interview with Brian Gardner.” *WPblogger*. Retrieved August 7, 2010. (`http://wpblogger.com/brian-gardner-gpl-interview.php`).

———. 2010b. “The GPL is Legal Castration.” *WPblogger*. Retrieved July 20, 2010. (`http://wpblogger.com/gpl-legal-castration.php`).

Cottrell, Allin. 1999. “Word Processors: Stupid and Inefficient.” *Allin Cottrell*. Retrieved August 22, 2010. (`http://ricardo.ecn.wfu.edu/~cottrell/wp.html`).

Daniels, Jessie. 2009. *Cyber Racism: White Supremacy Online and the New Attack on Civil Rights*. Lanham, Maryland: Rowman & Littlefield.

Darugar, Parand. 2006. “On Python, Ruby and Whitespace.” *Standard Deviations*. Retrieved August 22, 2010. (`http://parand.com/say/index.php/2006/12/11/on-python-ruby-and-whitespace/`).

David, Paul A. and Joseph S. Shapiro. 2008. “Community-based production of open-source software: What do we know about the developers who participate?” *Empirical Issues in Open Source Software* 20:364–398.

Debian Project. 2004. “Debian Social Contract.” *debian.org*. Retrieved January 4, 2010. (`http://www.debian.org/social_contract`).

Dery, Mark. 1995. *Flame Wars: The Discourse of Cyberculture*. Duke University Press.

DiBona, Chris and Sam Ockman, eds. 1999. *Open Sources: Voices from the Open Source Revolution*. Sebastopol, CA: O’Reilly.

DiMaggio, Paul, Eszter Hargittai, W. Russell Neuman, and John P. Robinson. 2001. "Social Implications of the Internet." *Annual Review of Sociology* 27:307–336.

Edgell, Penny, Joseph Gerteis, and Douglas Hartmann. 2006. "Atheists As "Other": Moral Boundaries and Cultural Membership in American Society." *American Sociological Review* 71.

FLOSSpols. 2006. "Gender: Integrated Report of Findings." *FLOSSpols*. Retrieved August 6, 2011. (`http://www.flosspols.org/deliverables/FLOSSPOLS-D16-Gender_Integrated_Report_of_Findings.pdf`).

Fourcade, Marion and Kieran Healy. 2007. "Moral views of Market Society." *Annual Review of Sociology* 33:285–311.

Free Software Foundation. 2009. "The Free Software Definition." *GNU Project*. Retrieved January 6, 2010. (`http://www.gnu.org/philosophy/free-sw.html`).

Fung, Archon and Erik Olin Wright, eds. 2003. *Deepening Democracy: Institutional Innovations in Empowered Participatory Governance*. London: Verso Books.

Gamson, Joshua. 1997. "Messages of Exclusion: Gender, Movements, and Symbolic Boundaries." *Gender and Society* 11:pp. 178–199. ISSN 08912432. (`http://www.jstor.org/stable/190542`).

Ghosh, R.A., R. Glott, B. Krieger, and G. Robles. 2002. "Free/Libre and Open Software: Survey and Study. Part IV: Survey of Developers." *Institute of Infonomics*.

Ghosh, Rishab Aiyer. 2005. "Understanding Free Software Developers: Findings from the FLOSS Study." In *Perspectives on Free and Open Source Software*, edited by Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani, chapter 2, pp. 23–45. The MIT Press. ISBN 0262062461.

Giddens, Anthony. 1987. *Social Theory and Modern Sociology*. Stanford University Press, 1 edition. ISBN 0804713561. (`http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0804713561`).

Gieryn, Thomas F. 1999. *Cultural boundaries of science : credibility on the line*. Chicago: University of Chicago Press.

GNU General Public License. 2007. “The Free Software Definition.” *GNU Project*. Retrieved January 6, 2010. (`http://www.gnu.org/copyleft/gpl.html`).

Goffman, Erving. 1989. “On Fieldwork.” *Journal of Contemporary Ethnography* 18:123–32.

Grimmelmann, James. 2005. “Ethical Visions of Copyright Law.” *Fordham Law Review* 77:2005–2037.

Gruber, John. 2008. “iPhone 3G.” *Daring Fireball*. Retrieved February 6, 2010. (`http://daringfireball.net/2008/10/iphone_3g`).

Guillen, Mauro F. and Sandra L. Suarez. 2005. “Explaining the Global Digital Divide: Economic, Political and Sociological Drivers of Cross-National Internet Use.” *Social Forces* 84:681–708.

Hangen, Nathan. 2010. “The GPL is Marxist.” *Nathan Hangen*. Retrieved July 20, 2010. (`http://nathanhangen.com/blog/mullenweg-gpl-marxist-utopia/`).

Hargittai, Eszter. 2010. “Digital Na(t)ives Variation in Internet Skills and Usus among Members of the "Net Generation".” *Sociological Inquiry* 80:92–113.

Hargittai, Eszter and Amanda Hinnant. 2008. “Digital Inequality: Differences in Young Adults’ Use of the Internet.” *Communication Research* 35:602–621. doi:10.1177/0093650208321782.

Healy, Kieran. 2006. *Last Best Gifts: Altruism and the Market for Human Blood and Organs*. Chicago: University of Chicago Press.

Hertzfeld, Andy. 2004. “Reality Distortion Field.” *Folklore.org: Macintosh Stories*. Retrieved February 6, 2010. (`http://folklore.org/StoryView.py?project=Macintosh&story=Reality_Distortion_Field.txt`).

Hessel, Andrew. 1999. “Open Source Biology.” In *Open Sources: Voices from the Open Source Revolution*, edited by Chris DiBona and Sam Ockman, chapter 18, pp. 281–296. Sebastopol, CA: O’Reilly.

Hewitt, Joe. 2010. “Android and Open Source.” *joehewitt.com*. Retrieved December 19, 2010. (`http://joehewitt.com/post/android-and-open-source/`).

Himanen, Pekka. 2001. *The Hacker Ethic*. New York: Random House.

Hitlin, Steven and Stephen Vaisey, eds. 2010. *Handbook of the Sociology of Morality*. New York: Springer.

Hodge, Nathan. 2010. "Disaster Relief 2.0: Tech Tools Help Focus Haiti Resources." *Wired*. Retrieved December 10, 2010. (`http://www.wired.com/dangerroom/2010/01/disaster-relief-20-haitis-virtual-surge/`).

Hoffman, Donna, Thomas P. Novak, and Ann Schlosser. 2000. "The Evolution of the Digital Divide: How Gaps in Internet Access May Impact Electronic Commerce." *Journal of Computer-Mediated Communication* 5.

Inc. 2008. "Matt Mullenweg." *Inc.* Retrieved July 29, 2011. (`http://www.inc.com/30under30/2008/profile/18-mullenweg.html`).

Jalkut, Daniel. 2009. "Getting Pretty Lonely." *Red Sweater Blog*. Retrieved December 17, 2010. (`http://www.red-sweater.com/blog/825/getting-pretty-lonely`).

Jaquith, Mark. 2010. "Why WordPress Themes are Derivative of WordPress." *Mark on WordPress*. Retrieved July 29, 2011. (`http://markjaquith.wordpress.com/2010/07/17/why-wordpress-themes-are-derivative-of-wordpress/`).

Kahn, David. 1996. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. New York: Scribner.

Kaplan-Moss, Jacob. 2008. "Building Django." *Colorado Software Summit*. Retrieved October 5, 2011. (`http://www.softwaresummit.com/2008/speakers/presentations/Kaplan-MossDjangoDesignDecisionsNotes.pdf`).

Keller, James. 1996. "Public access issues: An introduction." In *Public Access to the Internet*, edited by B. Kahin and J. Keller. MIT Press.

Kelty, Christopher M. 2008. *Two Bits: The Cultural Significance of Free Software*. Duke University Press.

Kim, Ryan. 2009. "Programming newbies make apps for iPhone." *San Francisco Chronicle*. Retrieved February 7, 2010. (`http://www.sfgate.com/cgi-bin/article.cgi?file=/c/a/2009/04/20/MN2O173593.DTL`).

Kincaid, Jason. 2009. "Facebook iPhone Dev Quits Project Over Apple Tyranny." *TechCrunch*. Retrieved February 7, 2010. (`http://www.techcrunch.com/2009/11/11/joe-hewitt-developer-of-facebooks-massively-popular-iphone-app-quits-the-project/`).

K.Mandla. 2010. "Maximalism is a better word." *Motho ke motho ka botho*. Retrieved August 27, 2010. (`http://kmandla.wordpress.com/2010/05/05/maximalism-is-a-better-word/`).

Kravets, David. 2009. "iPhone Jailbreaking Could Crash Cellphone Towers, Apple Claims." *Wired*. Retrieved February 20, 2010. (`http://www.wired.com/threatlevel/2009/07/jailbreak/`).

Kreiss, Daniel. 2011. "Open Source as Practice and Ideology: The Origin of Howard Dean's Innovations in Electoral Politics." *Journal of Information Technology and Politics* 8:367–382.

Krishnamurthy, Sandeep. 2006. "On the intrinsic and extrinsic motivation of free/libre/open source (FLOSS) developers." *Knowledge Technology Policy* 18.

Lakhani, Karim R. and Robert G. Wolf. 2005. "Why Hackers Do What They Do." In *Perspectives on Free and Open Source Software*, edited by Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani, chapter 1, pp. 3–21. The MIT Press. ISBN 0262062461.

Lamont, Michèle. 1994. *Money, Morals, and Manners: The Culture of the French and the American Upper-Middle Class*. Chicago: University of Chicago Press.

Lamont, Michèle and Virág Molnár. 2002. "The Study of Boundaries in the Social Sciences." *Annual Review of Sociology* 28:167–195.

Lande, Brian. 2007. "Breathing like a soldier: culture incarnate." *The Sociological Review* 55:95–108.

Lerner, Josh and Jean Tirole. 2005a. "Economic Perspectives on Open Source." In *Perspectives on Free and Open Source Software*, edited by Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani, chapter 3, pp. 47–78. The MIT Press. ISBN 0262062461.

———. 2005b. "The economics of technology sharing: Open source and beyond." *Journal of Economic Perspectives* 19:99–120.

Lessig, Lawrence. 2005. *Free Culture: The Nature and Future of Creativity*. Penguin (Non-Classics). ISBN 0143034650. (http://www.worldcat.org/isbn/0143034650).

———. 2008. *Remix: Making Art and Commerce Thrive in the Hybrid Economy*. Penguin Press HC, The. ISBN 1594201722. (http://www.worldcat.org/isbn/1594201722).

Levy, Steven. 2001/1984. *Hackers: Heroes of the Computer Revolution*. New York: Penguin Books.

Lewis, Kevin, Jason Kaufman, Marco Gonzalez, Andreas Wimmer, and Nicholas Christakis. 2008. "Tastes, Ties, and Time: A New Social Network Dataset using Facebook.com." *Social Networks* 30:330–342.

Lowe, Brian M. 2010. "The Creation and Establishment of Moral Vocabularies: Why Moral Vocabularies Matter." In *Handbook of the Sociology of Morality*, edited by Steven Hitlin and Stephen Vaisey, pp. 293–312. New York: Springer.

Manesh, Nasser. 2002. "Unix or UNIX?" *Unixica.com*. Retrieved January 9, 2010. (http://www.unixica.com/html/unixunix.html).

Markoff, John. 2006. *What the Dormouse Said: How the Sixties Counterculture Shaped the Personal Computer Industry*. New York: Penguin.

Martin, Steve and John Robinson. 2007. "The income digital divide: Trends and predictions for levels of Internet use." *Social Problems* 54:1–22.

Massengill, Rebekah P. and Amy Reynolds. 2010. "Moral Discourse in Economic Contexts." In *Handbook of the Sociology of Morality*, edited by Steven Hitlin and Stephen Vaisey, pp. 485–501. New York: Springer.

Matzan, Jem. 2005. "Distro review: The four-1-1 on Linspire Five-O." *Linux.com*. Retrieved June 15, 2010. (http://www.linux.com/archive/articles/113979).

McGowan, David. 2005. "Legal Aspects of Free and Open Source Software." In *Perspectives on Free and Open Source Software*, edited by Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani, chapter 1, pp. 361–391. The MIT Press. ISBN 0262062461.

Miles, Jr. Rufus E. 1978. "The Origin and Meaning of Miles' Law." *Public Administration Review* 38:399–403.

Moody, Glyn. 2001. *Rebel Code: The Inside Story of Linux and the Open Source Revolution*. New York: Basic Books.

Moon, Peter. 2007. "Stallman: If you want freedom don't follow Linus Torvalds." *PC World*. Retrieved June 19, 2010. (`http://www.pcworld.idg.com.au/article/195096/stallman_want_freedom_don_t_follow_linus_torvalds/`).

Mullenweg, Matt. 2009a. "Not Lonely At All." *ma.tt*. Retrieved October 5, 2011. (`http://ma.tt/2009/07/not-lonely-at-all/`).

———. 2009b. "Themes are GPL, too." *WordPress News*. Retrieved July 29, 2011. (`http://wordpress.org/news/2009/07/themes-are-gpl-too/`).

Nagel, Joane. 2001. "Racial, Ethnic and National Boundaries: Sexual Intersections and Symbolic Interactions." *Symbolic Interaction* 24:123–139.

Nissenbaum, Helen. 2004. "Hackers and the contested ontology of cyberspace." *New Media Society* 6:195–217. doi:10.1177/1461444804041445.

Ono, Hiroshi and Madeline Zavodny. 2008. "Immigrants, English Ability and the Digital Divide." *Social Forces* 86:1455–1479.

Open Source Initiative. 1999. "Open Source Definition." *opensource.org*. Retrieved January 4, 2010. (`http://www.opensource.org/osd.html`).

Patel, Nilay. 2011. "How Google controls Android: digging deep into the Skyhook filings." *This is my next*. Retrieved August 6, 2011. (`http://thisismynext.com/2011/05/12/google-android-skyhook-lawsuit-motorola-samsung/`).

Pateman, Carole. 1976. *Participation and Democratic Theory*. Cambridge: Cambridge University Press. ISBN 052129004X.

Pick, Michael. 2000. "Matt Mullenweg: WordPress and the GPL." *WordPress.tv*. Retrieved October 5, 2011. (`http://wordpress.tv/2009/10/13/matt-mullenweg-wordpress-gpl/`).

Pilgrim, Mark. 2004. "Freedom 0." *Dive into Mark*. Retrieved July 29, 2011. (http://diveintomark.org/archives/2004/05/14/freedom-0).

———. 2010. "Tinkerer's Sunset." *Dive into Mark*. Retrieved August 27, 2010. (http://diveintomark.org/archives/2010/01/29/tinkerers-sunset).

Poirier, Charline. 2010. "When users first encounter Ubuntu: six showstoppers." *Canonical Design*. Retrieved June 18, 2010. (http://design.canonical.com/2010/06/when-new-users-first-encounter-ubuntu-5-show-stoppers/).

Powell, Walter W. and Kaisa Snellman. 2004. "The Knowledge Economy." *Annual Review of Sociology* 30:199–220.

Ragin, Charles C. 1994. *Constructing Social Research*. Thousand Oaks, California: Pine Forge Press.

Raymond, Eric S. 2001. *The Cathedral and the Bazaar*. New York: O'Reilly. (http://www.catb.org/~esr/writings/cathedral-bazaar/).

———. 2003. *The Art of UNIX Programming*. Addison-Wesley Professional.

———. 2009. "The Economic Case Against the GPL." *Armed and Dangerous*. Retrieved October 5, 2011. (http://esr.ibiblio.org/?p=928).

Riley, Duncan. 2008. "More Hypocrisy from Mullenweg and WordPress with new themes jihad." *The Inquisitr*. Retrieved July 20, 2010. (http://www.inquisitr.com/11963/more-hypocrisy-from-mullenweg-and-wordpress-with-new-themes-jihad/).

Robb, John. 2007. *Brave New War: The Next Stage of Terrorism and the End of Globalization*. New York: Wiley.

Rothschild, Joyce and J. Allen Whitt. 1986. *The Cooperative Workplace: Potentials and Dilemmas of Organizational Democracy and Participation*. Cambridge University Press.

Rutkowski, Gil. 2010. "WordPress themes and plugins DO NOT inherit the GPL v2." *Flashing Cursor*. Retrieved August 6, 2011. (http://flashingcursor.com/wordpress/wordpress-themes-and-plugins-do-not-inherit-the-gpl-v2-152).

Sahana Software Foundation. 2010. *Sahana*. Retrieved December 10, 2010. (`http://www.sahanafoundation.org/`).

Salus, Peter H. 1994. *A Quarter Century of UNIX*. Boston, MA: Addison-Wesley.

Seltzer, Wendy. 2006. "Why Open Source Needs Copyright Politics." In *Open Sources 2.0: The Continuing Revolution*, edited by Chris DiBona, Danese Cooper, and Mark Stone, chapter 10, pp. 149–160. Sebastopol, CA: O'Reilly.

Shirky, Clay. 2008. *Here Comes Everybody: The Power of Organizing Without Organizations*. The Penguin Press, 1st edition. ISBN 1594201536.

Shuttleworth, Mark. 2010. "Comment 167 for bug 532633." *Ubuntu in Launchpad*. Retrieved June 15, 2010. (`https://bugs.launchpad.net/ubuntu/+source/light-themes/+bug/532633/comments/167`).

Smajda, Jon. 2011. "Open Source and the Moral Field of Computing." *Journal of Information Technology and Politics* 8:304–322. (`http://files.smajda.com/jon/papers/osmf.pdf`).

Smajda, Jon and Joseph Gerteis. 2012. "Ethnic communities and ethnic boundaries in a 'sauce-scented neighborhood'." *Sociological Forum* 27. (`http://files.smajda.com/jon/papers/northend.pdf`).

Somers, Margaret. 1994. "The Narrative Constitution of Identity: A Relational and Network Approach." *Theory and Society* 23:605–649.

Spencer, Dale C. 2009. "Habit(us), Body Techniques and Body Callusing: An Ethnography of Mixed Martial Arts." *Body & Society* 15:119–143. doi:10.1177/1357034X09347224.

Stallman, Richard M. 1999. "The GNU Operating System and the Free Software Movement." In *Open Sources : Voices from the Open Source Revolution*. O'Reilly. (`http://oreilly.com/catalog/opensources/book/stallman.html`).

———. 2001. "Free Software: Freedom and Cooperation." *GNU Project*. Retrieved January 6, 2010. (`http://www.gnu.org/events/rms-nyu-2001-transcript.txt`).

———. 2002. *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Free Software Society. (`http://www.gnu.org/philosophy/fsfs/rms-essays.pdf`).

Stephenson, Neal. 1999. "In the Beginning was the Command Line." Retrieved August 27, 2010. (`http://www.cryptonomicon.com/beginning.html`).

Stets, Jan E. 2010. "The Social Psychology of the Moral Identity." In *Handbook of the Sociology of Morality*, edited by Steven Hitlin and Stephen Vaisey, pp. 385–409. New York: Springer.

Surowiecki, James. 2005. *The Wisdom of Crowds*. Anchor. ISBN 0385721706.

Szabo, Gabor. 2010. "Comparing Perl and Python." *Blog of Gabor Szabo*. Retrieved August 22, 2010. (`http://szabgab.com/blog/2010/06/comparing-perl-and-python.html`).

Tapscott, Don and Art Caston. 1993. *Paradigm Shift: The New Promise of Information Technology*. McGraw-Hill.

Tapscott, Don and Anthony Williams. 2006. *Wikinomics: How Mass Collaboration Changes Everything*. Portfolio Hardcover. ISBN 1591841380.

Thomas, Matthew Paul. 2008. "Why Free Software has poor usability, and how to improve it." *Matthew Paul Thomas*. Retrieved June 15, 2010. (`http://mpt.net.nz/archive/2008/08/01/free-software-usability`).

Torvalds, Linus. 2001. *Just For Fun: The Story of an Accidental Revolutionary*. New York: HarperCollins.

———. 2007. "Re: [RFC] Convert builin-mailinfo.c to use The Better String Library." *Git Mailing List*. Retrieved August 27, 2010. (`http://lwn.net/Articles/249460/`).

van Rossum, Guido. 1998. "Open Source Summit Trip Report." *Linux Gazette*. Retrieved January 6, 2010. (`http://linuxgazette.net/issue28/rossum.html`).

VisionMobile. 2011. "Open Governance Index: Measuring the true opennes of open source projects from Android to WebKit." *VisionMobile*. Retrieved August 6, 2011. (`http://www.visionmobile.com/research.php#OGI`).

von Hippel, Eric. 2005a. *Democratizing Innovation*. The MIT Press. ISBN 0262002744.

———. 2005b. "Open Source Software Projects as User Innovation Networks." In *Perspectives on Free and Open Source Software*, edited by Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani, chapter 14, pp. 267–278. The MIT Press. ISBN 0262062461.

von Lohman, Fred. 2009. "Apple Says iPhone Jailbreaking is Illegal." *Electronic Frontier Foundation*. Retrieved February 20, 2010. (`http://www.eff.org/deeplinks/2009/02/apple-says-jailbreaking-illegal`).

Wacquant, Loïc. 2003. *Body and Soul: Notebooks of an Apprentice Boxer*. Oxford University Press.

———. 2005. "Carnal Connections: On Embodiment, Apprenticeship, and Membership." *Qualitative Sociology* 28:445–474.

———. 2009. "Habitus as Topic and Tool: Reflections on becoming a prizefighter." In *Ethnographies Revisited*, edited by Steven Kleinknecht William Shaffir, Antony Puddephatt. New York: Routledge.

Wareham, Richard. 2004. "The Command Line - The Best Newbie Interface?" *OS News*. Retrieved August 20, 2010. (`http://www.osnews.com/story/6282`).

Warner, Andrew. 2010. "Would WordPress Sue The Maker Of Thesis, A Leading WordPress Theme? – with Chris Pearson and Matt Mullenweg." *Mixergy*. Retrieved August 7, 2010. (`http://mixergy.com/chris-pearson-matt-mullenweg/`).

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

Weiner, Dave. 2009. "Hey Mike, I told you so." *Scripting News*. Retrieved August 2, 2009. (`http://www.scripting.com/stories/2009/08/01/heyMikeIToldYouSo.html`).

Wellman, Barry. 2001. "Computer Networks and Social Networks." *Science* 293:2031–2034.

Wherry, Frederick F. 2010. "The Sacred and the Profane in the Marketplace." In *Handbook of the Sociology of Morality*, edited by Steven Hitlin and Stephen Vaisey, pp. 147–161. New York: Springer.

Williams, Sam. 2002. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly. (`http://oreilly.com/openbook/freedom/`).

Winchester, Dan and Steven Hitlin. 2010. "The good, the bad, and the social." *Contexts* 9:40–44.

Woodford, Peter. 2004. “Open source medicine: cure for what ails the Third World?” *National Review of Medicine* 1 Retrieved January 4, 2010. (http://www.nationalreviewofmedicine.com/issue/2004/09_23/government_medicine02_17.html).

Zelizer, Viviana A. 1979. *Morals and Markets: The Development of Life Insurance in the United States.* New York: Columbia University Press.

———. 1985. *Pricing the Priceless Child.* Princeton, NJ: Princeton University Press.

Part IV

Appendices

Appendix A

The Hacker Habitus

Chapter Preview: In 2010, I was invited to submit a paper on the "hacker habitus" for a special issue on the topic of habitus. The special issue never materialized, and the paper was rejected as a standalone submission. I agree with the reviewers it is a bit rough and incomplete—particularly when evaluated as a standalone article. I set it aside to finish the rest of this dissertation as well as another paper (Smajda and Gerteis 2012).[1] *However, there's much I like about the paper, and I think it compliments the rest of this dissertation quite nicely. I did not find a way to fit it into the main body of the dissertation, but I believe it's worth including here as a complimentary piece to the main text.*

The paper is framed in terms of recent research and theory on the embodied nature of social behavior and cultural transmission. This approach finds a natural home in the study of obviously physical institutions such as boxing or policing. However, computer programming is one domain where physicality is apparently de-emphasized in place of textual, impersonal modes of communication and socialization. The paper attempts to illustrate how physical interaction with text-centric tools in programming is, in fact, a vehicle for the transmission of a shared, embodied culture of ethical and aesthetic values and norms of hacker culture. In particular, this paper investigates physical interaction with the Unix command line, text editors, and programming languages.

[1]Yes, I am totally citing that here.

A.1 Introduction

One of the strengths of the concept of habitus has been its utility in helping social scientists uncover and understand the subconscious, non-discursive, embodied aspects of social behavior and culture. Wacquant, the most prominent proponent of this characteristic of habitus, has advanced a "carnal sociology" that emphasizes an "incarnate approach" to studying social action, such as, in Wacquant's case, boxing (Wacquant 2003, 2005). Such an approach rejects the "animal symbolicum" of interpretive anthropology and symbolic interactionism (Wacquant 2009:8), and has inspired "flesh and blood" studies of "bodily crafts" such as military training (Lande 2007), bodybuilding (Bridges 2009), and mixed martial arts (Spencer 2009).

Software developers, it may seem, make an odd target for this kind of analysis. While Wacquant emphasizes the lack of explicit, written instructions involved in transmitting the "pugilistic habitus" to the apprentice boxer, hacker culture is heavily mediated through text: from source code and documentation to mailing lists, forums, IRC, and blogs. While face-to-face interactions and conferences do play a vital part in hacker culture (Coleman 2010), programmers are nonetheless depicted as "the quintessential digital subjects." If ever there were a group for whom "animal symbolicum" was an appropriate theoretical model, hackers would seem to be it.[2]

However, I will argue that a complete understanding of hacker culture should include an analysis of the embodied nature of hacking and the transmission of hacker culture. In particular, I will look at how physical interaction with the tools of the trade—the Unix command line, text editors, and programming languages—forms a central part of shared hacker experience and culture. Despite the image of the lone, physically inactive hacker hunched over a keyboard, I will argue that the tools hackers use are complicit in the transmission of hacker culture. The privileging, even fetishization, of plain text over rich text, the keyboard over the mouse, and lower levels of machine interaction over higher levels, inculcate a particular ethic and aesthetic about the production of software and culture more generally.

My investigation of the hacker habitus in this paper is admittedly incomplete. As a start, I am primarily interested in investigating how this "embodied" tradition of habitus research

[2]In popular culture, "hacker" has taken on more negative connotations in recent decades. In this paper, hacker refers to the original, positive meaning (see Levy (2001/1984), for example). For those interested in how these two, contradictory images of hackers came into being, Nissenbaum (2004) is a good starting point.

can help us understand a practice such as computing programming. Consequently, I have limited the scope of my inquiry to the tools programmers use to code. For example, the role of communication norms on forums, mailing lists, and IRC no doubt play a role in shaping the hacker habitus, though I will mostly skim over this topic here. The computing field also consists of a global industry of large corporations, small business, professional programmers, hobbyist hackers, and end users and consumers. This single paper cannot possible present a thorough account of this vast topic. However, this paper is part of a larger project on the moral and political culture of computing that aims to do just that (For example: Smajda 2011). This larger project is the result of 3 years of immersion in the software development community. In addition to systematically tracking blogs, forums, mailing lists, and public deliberation by participants across many open source software projects, I learned how to program and contributed to several open source projects myself.[3] This hands-on experience gave me a perspective on the art and science of programming that cannot be done justice by only focusing on, say, public deliberation by software developers. In this paper I will illustrate how the act of programming itself, and the tools with which this is done, are central to the production and reproduction of the *hacker habitus*.

A.2 Tools of the Trade

I will focus on the importance of three core tools in the hacker's toolkit: the Unix command line, the text editor, and programming languages. To those unfamiliar with these tools, the following subsections may at times feel a bit too much like a technical tutorial. I've tried to balance technical details with sociological analysis, though if at some points it feels like I err to far in the direction of technical detail, I encourage you to stick it out: the details I've chosen set up sociological points later in the paper.

A.2.1 The Command Line

For most people these days, using a computer means interacting with a graphical user interface (GUI). A GUI presents a collection of the now familiar graphical objects on screen like windows, a cursor, buttons, menus and scrollbars that all suggest the kinds of actions a user

[3]You can view much of this code—including a handful of plugins for the WordPress content management system, a browser-based interface for the todo.txt task management system, and more—at github.com/smajda.

can take. With a good GUI, a user doesn't have to memorize every action because the system provides visual cues at each step. In fact, a good GUI often requires no manual at all: the user can simply discover how the program works by clicking around. Consequently, GUIs are often easier to learn, and generally viewed as more intuitive.[4]

Figure A.1: The Unix Prompt

```
$ █
```

A command line interface (CLI), on the other hand presents the user with a simple, plain text prompt and cursor that looks something like Figure A.1. All interaction with the machine through a CLI is text-based: the user types out text commands, the computer processes those commands and returns with text output. To the casual observer, this seems anachronistic and unnecessary, but the command line, and in particular the Unix command line, remains a favored mode of interacting with the computer for hackers. Once accustomed to the command line, GUI's can appear wildly inefficient: why waste all those CPU cycles drawing colorful widgets on the screen just to do what you could have done by typing a few simple commands? This sentiment is colorfully expressed by Neal Stephenson, science fiction author and hacker, in his famous essay, *In the Beginning was the Command Line* (Stephenson 1999), by comparing Unix to the Hole Hawg drill:

> The Hole Hawg is a drill made by the Milwaukee Tool Company. If you look in a typical hardware store you may find smaller Milwaukee drills but not the Hole Hawg, which is too powerful and too expensive for homeowners. The Hole Hawg does not have the pistol-like design of a cheap homeowner's drill. It is a cube of solid metal with a handle sticking out of one face and a chuck mounted in another. The cube contains a disconcertingly potent electric motor. You can hold the handle and operate the trigger with your index finger, but unless you are exceptionally strong you cannot control the weight of the Hole Hawg with

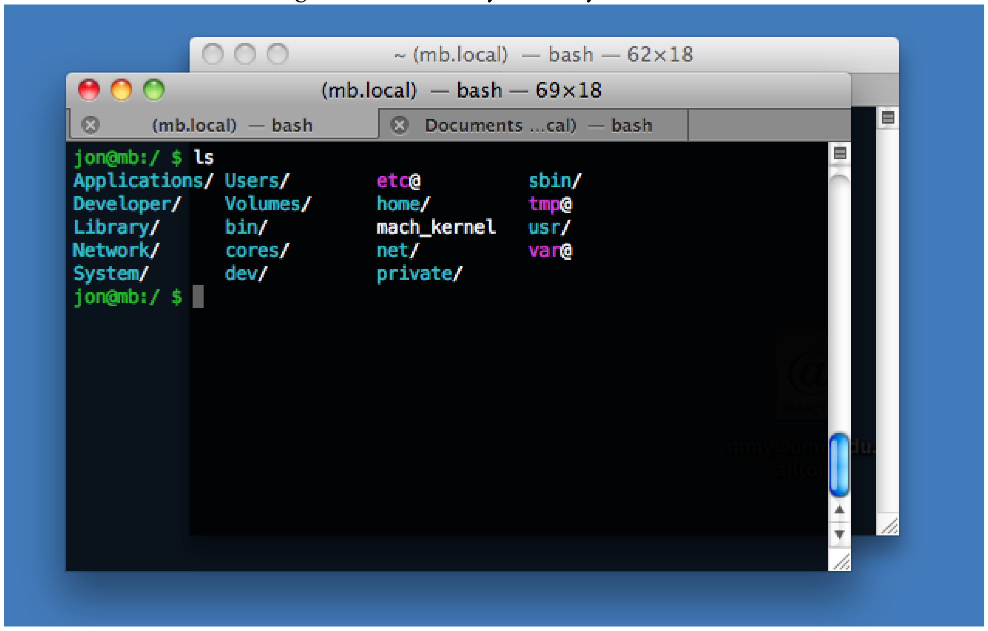
[4]Not everyone agrees: Wareham (2004).

> one hand; it is a two-hander all the way. In order to fight off the counter-torque of the Hole Hawg you use a separate handle (provided), which you screw into one side of the iron cube or the other depending on whether you are using your left or right hand to operate the trigger. This handle is not a sleek, ergonomically designed item as it would be in a homeowner's drill. It is simply a foot-long chunk of regular galvanized pipe, threaded on one end, with a black rubber handle on the other.
>
> [...]
>
> It is not hard to imagine what the world would look like to someone who had been raised by contractors and who had never used any drill other than a Hole Hawg. Such a person, presented with the best and most expensive hardware-store drill, would not even recognize it as such. He might instead misidentify it as a child's toy, or some kind of motorized screwdriver. If a salesperson or a deluded homeowner referred to it as a drill, he would laugh and tell them that they were mistaken—they simply had their terminology wrong.

The Hole Hawg, in other words, cultivates a very different orientation towards not just drills, but the act of construction; it's a component of the contractor's habitus versus the consumer's do-it-yourself, home improvement habitus. Stephenson goes on to argue that Unix is "the Hole Hawg of operating systems." For Unix hackers, expectations about the way an operating system should look, feel, and function—and why computers even exist, what they are for—are completely different than your average consumer-level computer user. While today's Unix-based systems (from Mac OS X to Linux) have all the eye candy interface elements we've come to associate with today's computers, saying you "know Unix" does not mean you know how to push around these buttons, checkboxes, and widgets with your mouse. To say you know Unix means you go beneath these sleek, ergonomic toys to the core of the Unix operating system: the command line. The Unix command line at the heart of any Mac or Linux computer today is fundamentally the same command line invented by Ken Thompson and Dennis Ritchie 40 years ago. Like the Hole Hawg, the Unix command line is built for power, not comfort or ease-of-use.

A more comfortable command line is possible, however. For example, it's possible to use relatively modern creature comforts such as text highlighting and windowing systems to go from a simple black screen with white text to something that looks more like Figure A.2:

Figure A.2: The Eye Candy Terminal



```
jon@mb:/ $ ls
Applications/ Users/       etc@         sbin/
Developer/    Volumes/     home/        tmp@
Library/      bin/         mach_kernel  usr/
Network/      cores/       net/         var@
System/       dev/         private/
jon@mb:/ $
```

multiple windows of terminal *emulators* within a graphical windowing system (here Mac OS X), a fancy prompt telling the user the name of the machine and current working directory, and text output color-coded by file type. In some circles, the modern creature comforts seen in Figure A.2 are dismissed as crutches—like putting fancy ergonomic handles on the Hole Hawg. For purists, there are large communities aiming for a stripped down, minimal Unix experience:

- One can install a window manager[5] such as "ratpoison," which, as its name implies, allows full operation of the computer without the use of that symbol of modern graphical computing: the mouse.
- There are still text-only web browsers under active development, such as elinks, that allow users to browse the web in a black and white, text-only environment.
- There are Linux users and communities dedicated to "pushing antiques and throwaway machines to perform in ways contemporary software and hardware manufacturers would probably prefer you didn't know about. Like writing blog posts from a 14-year-old

[5]The software that controls the drawing and manipulation of windows on your screen.

computer running modern, customized, bulletproof, rock-solid software that didn't cost a cent" (K.Mandla 2010).

- Other communities, such as minimallinux.com, are dedicated to finding the simplest tools possible for each task, such as CLI-based programs for note-taking or task management.

I do not want to imply that opting out of the creature comforts of the modern GUI is always a purely aesthetic decision or driven by social posturing. There are genuine advantages to manipulating computers through the command line. Additionally, the technical design of the command line is a factor in the social structure that has grown around it. As I described above, text is the universal interface for command line tools: text goes in, text comes out. As a result, it's very easy to write small, focused programs that interact with one another. During the development of Unix, this very feature is what enabled the fast growth of the operating system *and* the community around it. An operating system built of small, interlocking pieces that share a common interface facilitates the decentralized division of labor that drove the growth and popularity of Unix.

A.2.2 Text Editors

In an environment where everything is text, the text editor is of obvious importance. Most computer users are familiar with word processors, such as Microsoft Word. Text editors and word processors are not the same thing. Most word processors adhere to a What You See Is What You Get (WYSIWYG) philosophy. When sitting down at a computer to write with a word processor, we expect that when we want to italicize a word, we press a little "*I*" button in the toolbar and it italicizes the text we're typing *as we type it*. If we want to increase the margins of our printed page, we can increase the margins of the virtual page we're typing on. If we add a footnote, the virtual page on the screen puts a little footnote right where it'll be on the page that we print. In other words, the screen aims to be a simulation of the final printed page. Text editors do none of this. Just as opting for a CLI in place of a GUI appears anachronistic and backwards, so does working with *plain text* instead of the *rich text* of WYSIWYG word processors. Yet the combination of plain text edited in a text editor is the way in which virtually all the software we run begins its life.

In the Unix world, there are traditionally two big text editors: `vi` and `emacs`. I'm a `vi` (pronounced *vee-eye*) user, so I'll focus on vi, which has a particularly nasty reputation for being hard to learn.[6] Vi is a *modal* editor, making it even more of a throwback than a normal plain text editor. Rather than relying on menus or control key shortcuts to perform editing operations, the entire keyboard is switched into either Insert mode (for writing) or Command mode (for getting around and editing). It makes editing text a bit like playing a video game and requires you to learn an entire suite of keyboard gestures to edit a file. If you want to delete the current word, you type (in command mode) `dw`. If you want to copy the current line and paste a copy below, type `yy` followed by `p`. This seems crazy until you figure out you can chain these editing moves together into complex (and yes, *fun*) editing moves. Say you've got a compound sentence and you want to rewrite the part between the comma and the period (type `f,ct.`) or you want to find every occurrence of "Vi" and change it "Vim" (type `:%s/Vi/Vim/g`). Vi, at least traditionally, has no menus or toolbar of buttons to assist the user. For example, Figure A.3 shows this paper opened in vi. As with the Unix command line, the user simply has to learn this collection of commands to interact with the program.

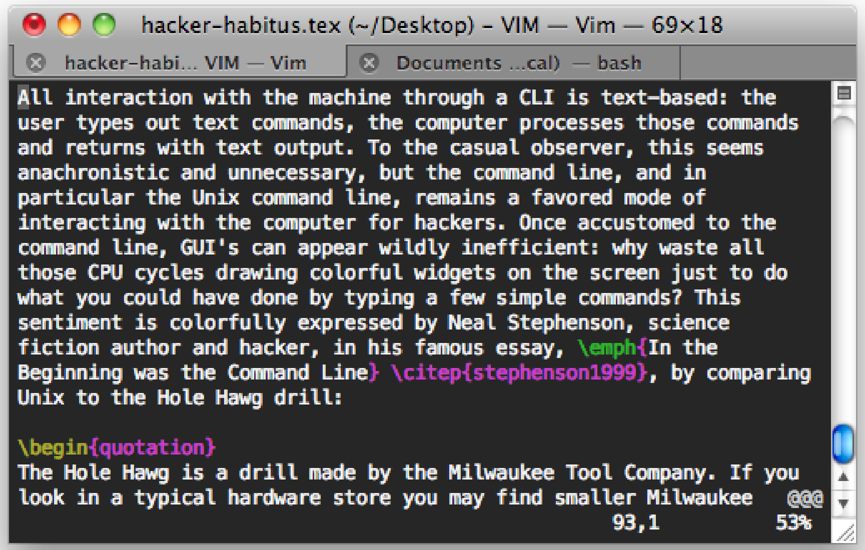
I provide these details to illustrate a few points about using `vi`, or any text editor like it for that matter. As with the Unix command line, there is a rather steep initial learning curve, though there is great power once you crack through this initial usability barrier: despite being decades old, programmers continue to return to the classic text editors because, while esoteric, they are very good at editing text.

Also, as with the command line, driving the keyboard in vi becomes second-nature over time. Acts such as moving up and down lines of text (with `j` and `k`) become like "native" actions, hard-coded into the gestural repertoire of a vi user's hands.

As we also saw with the command line vs. the GUI, there is an attitude that the rich text environments used by most computer users are, in the words of one oft-cited polemic against word processers, "stupid and inefficient" (Cottrell 1999). On this view, text editing is precise and efficient. Fixed width fonts are only uncomfortable to those spoiled by the print-like proportional width fonts used by word processors that "dumb down" the computing experience, locking the user into complicated, slow, expensive software peddled by corporations like Microsoft.

[6]"Vi" is the classic editor, while "Vim" (for "Vi Improved") is a modernized version which is what is actually installed on most modern systems. I'll use the two interchangably.

Figure A.3: Using vi



```
All interaction with the machine through a CLI is text-based: the
user types out text commands, the computer processes those commands
and returns with text output. To the casual observer, this seems
anachronistic and unnecessary, but the command line, and in
particular the Unix command line, remains a favored mode of
interacting with the computer for hackers. Once accustomed to the
command line, GUI's can appear wildly inefficient: why waste all
those CPU cycles drawing colorful widgets on the screen just to do
what you could have done by typing a few simple commands? This
sentiment is colorfully expressed by Neal Stephenson, science
fiction author and hacker, in his famous essay, \emph{In the
Beginning was the Command Line} \citep{stephenson1999}, by comparing
Unix to the Hole Hawg drill:

\begin{quotation}
The Hole Hawg is a drill made by the Milwaukee Tool Company. If you
look in a typical hardware store you may find smaller Milwaukee   @@@
                                                 93,1          53%
```

A.2.3 Programming Languages

Continuing our trip down the plain text rabbit hole of hacker culture, a brief look at programming languages is in order. Obviously, as with every other stop on our journey so far, no knowledge of programming languages is expected, nor will much be taught here. However, I've selected a few case studies to illustrate how programming languages become entangled in debates over the definition of good programming and good taste and skill among hackers.

As we saw with command line and text editors, there is a trend in hacker culture towards placing greater value on "lower level" programming languages. The trend in programming languages has been towards higher and higher levels of abstraction. For example, in the old days, coding meant writing in assembly language. (Well, in the old, old days, it meant punching a series of holes in cards and feeding those into machines.) Then, along with Unix, Dennis Ritchie invented C: a higher-level language that abstracted away machine-specific details, allowing one program to compile for multiple platforms. Still later, extensions of C such as C++ and C# added ever more abstracted layers on top of C, and so-called "scripting languages" (a term sometimes used with derision) such as Perl, PHP, and Ruby were created for even more lightweight, portable applications. In general, status within the hacker community corresponds to the level at which you work—the lower the better.

For example, Linus Torvalds, creator and coordinator of the Linux kernel, explains his choice of the lower-level C over C++ (Torvalds 2007):

> C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it, to the point where it's much much easier to generate total and utter crap with it. Quite frankly, even if the choice of C were to do *nothing* but keep the C++ programmers out, that in itself would be a huge reason to use C.
>
> . . . the only way to do good, efficient, and system-level and portable C++ ends up to limit yourself to all the things that are basically available in C. And limiting your project to C means that people don't screw that up, and also means that you get a lot of programmers that do actually understand low-level issues and don't screw things up with any idiotic "object model" crap.

Differences in syntax between languages—which, on the one hand, may be purely technical design decisions—also become embroiled in debates about values and ideals. For example, the Perl programming language offers programmers a lot of discretion in how they choose to write code, and Perl famously has "There's more than one way to do it" as a motto. On the other hand, another language, Python, has as a motto "There should be one—and preferably only one—obvious way to do it." As much as this may seem to be a mere technical issue, culturally this difference in design philosophy is interpreted and extrapolated to reveal differences in how members of the Perl and Python communities think about themselves, programming, and more. For example, Python has stricter rules about breaking up blocks of code with linebreaks and whitespace compared to Perl, which uses C-like braces. In a simple illustration of this, here is an "if" conditional in Perl:

```
if ($x == 7) {
   print 'x is 7';
}
```

And Python:

```
if x == 7:
   print 'x is 7'
```

This seems like a trivial difference between the two languages,[7] but programmers have extrapolated a range of social and psychological characteristics of Perl and Python users from this difference. For example, a non-Python programmer on Python:

> If they're this fascist about whitespace, just imagine what the rest of the language must look like![8] (Darugar 2006)

And a Perl programmer equates Python's relatively strict syntax rules with an affinity for centralized government:

> People who prefer more centralized governing will tend to use Python and think that it is the right way to do things. Seeing people use Perl makes them nervous or even aggressive. (Szabo 2010)

Much of this is usually done in a playful, tongue-in-cheek manner, but that doesn't refute the main point I'm trying to make here: that the technical design of a plain text programming language is interpreted in terms of what it implies about the social, psychological, and cultural features of those who design and use it. Programming languages and environments are not viewed as value-neutral tools for merely producing software, but as tools that embody and cultivate a particular kind of rationality and aesthetic that one must adopt in order to properly use. In the case of Python vs. Perl, the technical design of the language represents competing visions of what it means to be creative as a programmer and how to collaborate with others. In the quote by Linus Torvalds on C and C++ above, there is a technical argument (some things, like kernels, are best done with some tools and not others), but this argument also taps into the belief that the level of abstraction in a language has a negative correlation with intelligence and competence. This belief resonates with the principles underlying the design of programming tools and it can also function as a significant social barrier for participation by less experienced users (a bad or good thing depending on your viewpoint).[9]

[7]And my admittedly oversimplified example doesn't really convey the full implications of the design philosophies behind the two languages... but this is probably, as we say, beyond the scope of this paper.

[8]In fairness: the next sentence is "Now that I've used it for a little while, I must say I'm converted. Whitespace is the way to go."

[9]The PHP language is another good example of this. PHP is a popular first language for web developers and its

A.3 The Feeling of Hacking

As I mentioned earlier, my decision to focus on tools such as the vi editor and the Unix command line is informed by my personal experience. I did not grow up programming. In fact, it was only as a graduate student—through a research assistantship with our college's IT department—that I was introduced to these tools. As an assistant system administrator for our college's Unix research and web servers, I had to learn my way around the Unix command line, vi, and scripting in bash (the default Linux shell environment). I become hooked on these tools. Unlike the clunky, slow, crash-prone experiences I kept having with word processors and other GUI applications, Unix and vi were fast, elegant (once you grasped the basics), and flexible. But more than that, they *felt* different, more direct and *intimate* somehow. I soon realized I could use these tools for more and more of my computing tasks: for my academic writing (by using vi and LaTeX, a plain-text markup language for typesetting academic papers), to let me build and customize websites, and to automate and customize my own personal computing environment. In other words, I learned the tools first and only then realized all the cool things I could do with them.

This may seem backwards. One could imagine a more conventional, instrumental path to a career in programming would look more like this: strong analytical or mathematical skills, along with encouragement from teachers or advisors, lead a young person to the realization that programming could be a pretty good career. Following this realization, they set out to learn how to use the tools that enable them to program. However, my "play with the tools first" experience is not unusual and many hackers have similar stories. For example, Mark Pilgrim (2010) describes his first programming experience with an Apple][e:

> As it happens, this computer came with the BASIC programming language pre-installed. You didn't even need to boot a disk operating system. You could turn on the computer and press `Ctrl-Reset` and you'd get a prompt. And at this prompt, you could type in *an entire program*, and then type `RUN` and it would motherfucking run.
>
> I was 10. That was 27 years ago, but I still remember what it felt like when I realized that you — that *I* — could get this computer to do *anything* by typing the

wide availabiity and accessibility lead to it being classified as both a "working man's language" but also a language plagued by a community of less experienced, less capable programmers.

> right words in the right order and telling it to `RUN` and it would motherfucking run.

Citing the *feeling* of programming, the physical and emotional thrill of making text *do things* is a common theme in hacker autobiographies. Linus Torvald's autobiography (2001) is titled "Just for Fun." A famous quote by Richard Feynman jokingly calls this a disease:

> There is a computer disease that anybody who works with computers knows about. It's a very serious disease and it interferes completely with the work. The trouble with computers is that you 'play' with them!

From this perspective, the spartan interface of programmers' text editors, and the anachronistic, utilitarian asceticism of the command line contribute to the feeling of hacking, the thrill of being just a little bit "closer to the metal." Car enthusiasts prefer manual transmissions, professional contractors prefer the Hole Hawg, and hackers prefer the command line and low-level programming languages.

A.4 Habitus and Hacking

Throughout this paper, I've been struggling to balance two impulses. On the one hand, I confess to "going native" to a certain extent. I am still a relative "newbie," though I use, and enjoy, software like vi and the Unix command line everyday. As a "native" it is tempting to interpret the central role of tools such as the Unix command line and text editors in purely instrumental terms: these are simply the best tools for the job, and people who need power and efficiency will naturally learn, and enjoy using, them.

But from a sociological perspective, this picture is more complicated and the motives maybe not so virtuous. Hackers form an expert culture, and the fetishization of obscure programming tools and techniques, not to mention brusque idioms such as "RTFM" ("Read the Fine Manual"), serves to lock out newcomers and reinforce the status of established hackers. All this macho talk about being "closer to the metal" and boundary work around what constitutes a "true hacker" is clearly gendered in important ways as well. (Geek culture generally has its own peculiar form of masculinity that deserves more attention than I can adequately address in an aside in this paper.) To put this in Bourdiuean language: learning how to use tools like vi and the Unix command line constitutes cultural capital within hacker culture. As such, those who have access to these tools, the time to learn them, and the social connections to encourage their use, occupy a privileged position.

However, as far is this "cultural capital" position as stated so far is concerned, the content of the tools and knowledge themselves are irrelevant. If it wasn't the Unix command line functioning as privileged knowledge, it'd be something else, and the result would be the same. In other words, this involves no accounting for the experience of actually using these tools. This is where habitus directs us in a useful direction and the preceding discussion of the "tools of the trade" can help us out. First, the shape and feel of the tools matter. The feeling of interacting with the tools often serves as an entry point into programming, and the design of these tools embody particular values and ways of thinking. The physicality of this experience appears to be rather limited—typing at a keyboard while staring at a screen—and the sociality appears limited as well. But the values that are transmitted through these tools—such as "power" and "efficiency"—are not non-social. Ideals about power and efficiency drive the judgments hackers make about a range of things: from comparing users of competing languages and text editors to the way they think about the larger public that they may even be writing software for.

The take-home point for scholars interested in habitus and the body is that investigations into the physicality of habitus development and transmission need not be limited to extreme, perhaps novel, cases of overt physical action like boxing. The act of interacting with a physical machine, and even the metaphoric tools and objects represented on a screen by software, can carry a great deal of information about ways of thinking, feeling, and acting.

Appendix B

Scripts

B.1 Elinks script

One of the ugliest shell scripts you will ever see. Only bested by the one that follows.

```
#!/bin/sh
workpath="~/diss/tmp"
datapath="~/diss/data"
newdatapath="~/diss/data/todo"

#delicious:
deluser=''
delpw=''
delurl="https://api.del.icio.us/v1/posts/add?"

# check for flags
notesOnly='no' # just ignore this
while [ "${1:0:1}" = "-" ]; do
    case "$1" in
        '-n')
            notesOnly='yes'
            echo "NOTE: Only an info file & bookmark will be created."
        ;;
    esac
shift
done
url="$@"

# create refname from url & remove www. & .tld’s
refname=‘echo "$url" | awk -F "/" ’{print $3}’ | \
sed ’s/^www.//;s/.org$//;s/.net$//;s/.co.uk$//;s/.com$//’‘

# append unique number to refname
counter=1
newrefname=$refname’’$counter
while [ -f $datapath/*/"$newrefname".data ]; do
    counter=$((counter + 1))
    newrefname=$refname’’$counter
done
refname=$newrefname

# add extension for reffile
reffile=$refname.data

# grab url to html w/ curl
curl "$url" > $workpath/$refname.html

# how to pull title from html dump and read to $doctitle
doctitle=‘awk -F "</*[Tt][Ii][Tt][Ll][Ee]>" ’/\<[Tt][iI][Tt][Ll][Ee]\>/ {print $2}’ \
$workpath/$refname.html‘
doctitle=$refname" "$doctitle

echo "======Summary======"
echo "URL: $url"
echo "Title: $doctitle"
# offer to modify the title
echo "To keep title, hit return. To change, type new title now. (Ignore refname)"
    read newdoctitle
    if [ -z "$newdoctitle" ]; then
        echo " - Ok, title is still $doctitle"
    else
        doctitle=$refname" "$newdoctitle;
        echo " - Ok, title is now $doctitle"
    fi
# add tags
echo "Tags? (separate with spaces)"
read tags
echo $tags | sed 's/,//g' # remove commas I inevitably forget to not use
echo "Ok, Tags: $tags"


#### Ok, now create data file with elinks
# add "@@" to each tag to get @@tags in data file
dtags=`for i in $tags; do echo $i | awk '{print "@@"$1}'; done`
echo "Creating .data file..."
echo $dtags >> $newdatapath/$reffile
echo "URL: "$url"" >> $newdatapath/$reffile
echo "TITLE: "$doctitle"" >> $newdatapath/$reffile
echo "DATE PULLED: `date`" >> $newdatapath/$reffile
echo " "
# now do elinks -dump (as long as -n flag is used)
if [ $notesOnly != yes ]; then
    echo "Elinks is dumping..."
    elinks -dump "$workpath/$refname.html" >> $newdatapath/$reffile
fi
```

```
#### now post to getboo or delicious
#--data extended="Bar" \
#echo "Posting to getboo.."
#curl --user "$gbuser":"$gbpw" \
echo "Posting to delicious.."
curl --user "$deluser":"$delpw" \
--data url="$url" \
--data description="$doctitle" \
--data tags="$tags" \
$delurl
##$gburl

echo "Done."
```

B.2 Search script

Phew.

```
#!/bin/sh
### set variables
dataDir="~/diss/data"
sourceList=$dataDir/source-list.txt
cd $dataDir

### give help for empty command
if [ -z $1 ]; then
    echo "Usage: $0 [ -s | -d | -n ] PATTERN ARGUMENTS"
    echo "Options:"
    echo "          -s,             search source-list"
    echo "          -d,             search data"
    echo "          -n,             search notes"
    echo "Arguments added at the end get passed to grep"
    exit 1
fi

#### Grab options
while [ "${1:0:1}" = "-" ];do
    case "$1" in
        '-s')
            sourcePattern=$2
        ;;
        '-d')
            dataPattern=$2
        ;;
        '-n')
            notesPattern=$2
        ;;
        '-h')
            helpRequest=$2
        ;;
    esac
shift
shift
done
# everything left over can be used as $1 or $@

#will use this to pass options to grep later
if [ -z "$@" ]; then
  extraArgs="-H"
  else
    extraArgs="$@"
    extraArgs=`for i in $extraArgs; do echo -$i" "; done | tr -d '\n'`
    echo "$extraArgs"
fi

#### do something with options

## if -s is used and -d and -n are not, search source-list.txt
if [[ -n "$sourcePattern" && -z "$dataPattern"  && -z "$notesPattern" ]]; then
    awk -F ";" '/'$sourcePattern'/ {print $1 ", "$2}' $sourceList
fi

## if -d is used, first with -s, then without -s, search *.data
if [ -n "$dataPattern" ]; then
  if [ -n "$sourcePattern" ]; then
    sourcename=`awk -F ";" '/'$sourcePattern'/ {print $1}' $sourceList`
    for dirname in $sourcename; do
        grep --color -Hni $extraArgs "$dataPattern" $dirname/*.data
    done
  else
    for dirname in . ; do
      grep --color -Hni $extraArgs "$dataPattern" ./*/*.data
    done
  fi
fi

## if -n is used, first with -s, then without -s, search *.notes
if [ -n "$notesPattern" ]; then
  if [ -n "$sourcePattern" ]; then
    sourcename=`awk -F ";" '/'$sourcePattern'/ {print $1}' $sourceList`
    for dirname in $sourcename; do
      grep --color -Hni $extraArgs "$notesPattern" $dirname/*.notes
    done
  else
    for dirname in . ; do
      grep --color -Hni $extraArgs "$notesPattern" ./*/*.notes
    done
  fi
fi
```