Introduction

Joseph Feller, Brian Fitzgerald, Scott Hissam, and Karim R. Lakhani

What This Book Is About

Briefly stated, the terms "free software" and "open source software" refer to software products distributed under terms that allow users to:

- Use the software
- Modify the software
- Redistribute the software

in any manner they see fit, without requiring that they pay the author(s) of the software a royalty or fee for engaging in the listed activities. In general, such terms of distribution also protect what the publishing world calls the "moral right" of the software's author(s) to be identified as such. Products such as the GNU/Linux operating system, the Apache Web server, the Mozilla Web browser, the PHP programming language, and the OpenOffice productivity suite are all well-known examples of this kind of software.

More detailed, formal definitions for the terms *free* and *open source* are maintained—and vigilantly watch-dogged—by the Free Software Foundation (FSF)[1] and Open Source Initiative (OSI).[2] However, the definitions are substantively identical, and the decision to use one of these terms rather than the other is generally ideological, rather than functional; the FSF prefers the use of a term that explicitly refers to freedom, while the OSI believes that the dual meaning of the English word "free" (*gratis* or *libertas*) is confusing, and instead prefers the emphasis on the availability and modifiability of source code.[3] In Europe the French-English construct *libre software* has been widely adopted to unambiguously capture the connotation intended by the FSF.[4]

Free and open source software (F/OSS), however, is more than a set of terms of distribution. F/OSS is also—or, perhaps, primarily—a collection of

tools and processes with which people create, exchange, and exploit software and knowledge in a manner which has been repeatedly called "revolutionary."

Revolutions are a lot like caterpillars—they don't grow old. Either they die young, or they undergo metamorphosis into something quite different. Successful caterpillars become butterflies and successful revolutions replace, or at least transform, the status quo. What is the status of the F/OSS revolution? Has it successfully transformed the software industry? Other industries? Governments and societies? Or, is the revolution still in "chrysalis," with the great change to come tomorrow? Or, has the revolution already died young? Or is it, perhaps, doomed to do so?

In the broadest sense, this book was written to address these questions.

Perspectives on Free and Open Source Software

"In the broadest sense" won't get you very far, though, so we'll be a bit more precise. The earliest research and analysis on F/OSS emerged from within:

- The F/OSS community itself (including the writings of Richard M. Stallman and Eric S. Raymond)
- The technology press (for example *Wired* magazine, O'Reilly and Associates)
- The software engineering research community (for example the ACM and IEEE)

It didn't take long, however, for a substantial and well-rounded literature to emerge—one addressing F/OSS as not only a software engineering phenomenon, but as psychological, philosophical, social, cultural, political, economic, and managerial phenomena as well. The bibliography of this book[5] is testament to the variety and richness of this scholarship.

We wanted this book to bring together, under one roof, provocative and exemplary research and thinking from people within a number of different academic disciplines and industrial contexts. Specifically, we've gathered together work from many of the leading F/OSS researchers and analysts and organized them into five key "perspectives" on the topic. These parts are:

Part I. Motivation in Free/Open Source Software Development
Part II. The Evaluation of Free/Open Source Software Development
Part III. Free/Open Source Software Processes and Tools
Part IV. Free/Open Source Software Economic and Business Models
Part V. Law, Community and Society

Next, we describe each of these parts, offering short summaries of the chapters and suggesting key questions that the reader might bear in mind.

Part I: Motivation in Free/Open Source Software Development

Many first-time observers of the F/OSS phenomenon are startled by the simple fact that large numbers of highly skilled software developers (and users) dedicate tremendous amounts of time and effort to the creation, expansion, and ongoing maintenance of "free" products and services. This *seemingly* irrational behavior has captured the attention of reflective F/OSS community participants and observers.

The three chapters in Part I seek to better describe and understand the motivations of individuals who participate in F/OSS activities.

Lakhani and Wolf (chapter 1) report that the largest and most significant determinant of effort (hours/week) expended on a project was an individual sense of creativity felt by the developer. They surveyed 684 developers in 287 F/OSS projects on SourceForge.net and found that more than 60 percent rated their participation in the projects as the most (or equivalent to the most) creative experience in their lives. Respondents expressed a diverse range of motivations to participate, with 58 percent of them noting user need for software (work and non-work-related) as being important. Intellectual stimulation while coding (having fun), improving programming skills, and an ideological belief that software should be free/open were also important reasons for participating in a F/OSS project. The authors' analysis of the data shows four distinct clusters (approximately equal in size) of response types:

1. Those that expressed enjoyment and learning as primary motivators
2. Those that simply need the code to satisfy non-work-related user needs
3. Those that have work-related needs and career concerns
4. Those that feel an obligation to the community and believe that software should be free/open

These findings indicate an inherent source of strength within the F/OSS community. By allowing individuals with multiple motivation types to coexist and collaborate, the F/OSS community can and does attract a wide range of participants. Individuals can join for their own idiosyncratic reasons, and the F/OSS community does not have to be overly concerned about matching motivations to incentives.

Ghosh (chapter 2) presents a study conducted for the European Union of more than 2,700 F/OSS developers, and reports that more than 53

percent of the respondents indicated "social" motivations to join and continue in the community. The single most important motivation was "to learn and develop new skills." About 31 percent of the respondents noted career and monetary concerns, 13 percent indicated political motivations, and 3 percent had product requirements. Contrary to many altruism-based explanations of participation, Ghosh reports that 55 percent of respondents note "selfish" reasons to participate; that is, they state that they take in more than they contribute. Interestingly, he finds no difference in participation levels in projects between those that are motivated by social concerns and those that are motivated by career/monetary concerns.

Ghosh's study also showed that a majority of the developers are male, and that more than 60 percent are under age 26. Surprisingly (given the nerdish stereotypes prevalent in the mainstream view of F/OSS developers), more than 58 percent of the developers indicated having "significant other" partners with a large fraction (40 percent) living with their partners. About 17 percent of the respondents also indicated having at least one child.

Finally, chapter 3 presents a modified version of Lerner and Tirole's 2002 *Journal of Industrial Economics* paper, "Some Simple Economics of Open Source," one of the most widely cited papers in the F/OSS research literature. Lerner and Tirole employ a simple economic rationale of cost and benefit in explaining why developers choose to participate in F/OSS projects. As long as benefits exceed costs, it makes rational economic sense for a developer to participate in a project. Costs to the developers are defined mainly as opportunity costs in time and effort spent participating in creating a product where they do not get a direct monetary reward for their participation. Additional costs are also borne by organizations where these developers work if they are contributing to F/OSS projects during work hours.

Lerner and Tirole propose that the net benefit of participation consists of immediate and delayed payoffs. Immediate payoffs for F/OSS participation can include meeting user needs for particular software (where working on the project actually improves performance) and the enjoyment obtained by working on a "cool" project. Delayed benefits to participation include career advancement and ego gratification. Participants are able to indicate to potential employers their superior programming skills and talents by contributing code to projects where their performance can be monitored by any interested observer. Developers may also care about their reputation within the software community, and thus contribute code to

earn respect. In either case, delayed payoffs are a type of signaling incentive for potential and actual contributors to F/OSS projects.

Part II: The Evaluation of Free/Open Source Software Development

Part I asked "Why do they do it?"; Part II asks "Was it worth it?" In this section, we seek to address a wide range of issues related to evaluating the quality—security, reliability, maintainability, and so on—of both the F/OSS process and its products. Both pro- and anti-F/OSS rhetoric has too often been characterized by grandstanding and FUD[6] flinging. We are confident, then, that the chapters in this section meet some very real needs in both the academic and practitioner communities for objective, empirically grounded assessment.

Glass takes up this theme (the need for objectivity and sobriety) in chapter 4. He positions himself (with great ease and familiarity, it would seem) in front of what he calls the "steamroller" of unexamined hype. Glass raises a wide range of claims about F/OSS, regarding the talent of F/OSS community members, the security and reliability of the software, the sustainability of F/OSS economic and business models, amongst other issues. It is a provocative chapter, and we began Part II with it knowing it would wake you up and sharpen your wits. While you might not agree with all of Glass's arguments, his one overarching claim is irrefutable: if we are to understand and benefit from the F/OSS phenomenon, we cannot do so without robust research and hard evidence.

Fitzgerald (chapter 5), while not quite in front of the steamroller, is at least on the construction site. Drawing on a wide range of research and F/OSS writings, Fitzgerald articulates a number of what he calls "problematic issues," arising from software engineering, business, and sociocultural perspectives. These issues include the scarcity of developer talent (questions of motivation aside), the potentially negative effects of the modularity that characterizes many F/OSS products, the problems with "porting" the F/OSS process into sector-knowledge-intensive vertical software domains, and the churn caused by changing project (or even movement) leadership.

Rusovan, Lawford, and Parnas (chapter 6) change our tack slightly, moving away from the broader and more discursive tone of chapters 4 and 5. Instead, they focus on a single, concrete example, the findings from applying experimental software inspection techniques (Parnas 1994b) to a particular part of the TCP/IP implementation in GNU/Linux. Although they caution against resting an evaluation of the F/OSS process on a single

investigation, they do assert that the Linux ARP code was revealed to be "poorly documented," the interfaces "complex," and the module needlessly reliant on "what should be internal details of other modules." Their study points importantly to the need for elegant design and effective documentation in all software, even in the wilds of the "bazaar."

Neumann (chapter 7) in many ways echoes the implied challenges of the previous chapter—arguing that F/OSS is not inherently "better" than proprietary software, but that it has the potential to be. He points to, and briefly summarizes, the dialog that emerged from the 2000 IEEE Symposium on Security and Privacy, and concludes that F/OSS presents us with the opportunity to learn from mistakes which we should have learned from years ago.

Anderson (chapter 8) elaborates considerably on the issues raised by Neumann. Anderson walks the reader through the logic and formulae which demonstrate that releasing a system as F/OSS (thus opening the source code to public scrutiny) enables an attacker to discover vulnerabilities more quickly, but it helps the defenders exactly as much. He goes on to elaborate on the various, specific situations that may cause a break in the potential symmetry between proprietary and F/OSS products. The balance "can be pushed one way or another by many things," he argues, and it is in these practical deviations from the ideal that "the interesting questions lie."

Finally, Weinstock and Hissam (chapter 9) address a wide range of perceptions and "myths" related to the F/OSS phenomenon, and present data gathered in five case studies: the AllCommerce Web store in a box, the Apache HTTP server, the Enhydra application server, NAIS (a NASA-operated Web site that switched from Oracle to MySQL), and Teardrop (a successful Internet attack affecting both F/OSS and proprietary systems). They conclude that F/OSS is a viable source of components from which to build systems, but such components should not be chosen over other sources simply because the software is free/open source. They caution adopters not to embrace F/OSS blindly, but to carefully measure the real costs and benefits involved.

Part III: Free/Open Source Software Processes and Tools

Software engineering (SE) is a very young field of study. The first computer science department (in the United States) was established in just 1962 (Rice and Rosen 2002) and it wasn't until after a NATO conference on the "software crisis" in 1968 that the term "software engineering" came into

common use (Naur and Randall 1969; Bauer 1972), and the first degree program for software engineers in the United States wasn't established until 1978 at Texas Christian University.

Software engineering is more than just "coding," it is applying "a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" (IEEE 1990); it is also the engineering of software to meet some goal, and to see that the constructed software operates over time and that it is maintained during its expected life.[7]

Such definitions of software engineering have led the way to a plethora of processes, paradigms, techniques, and methodologies, all with the goal of helping to make the process of engineering correct software repeatable and addressing the concerns raised at the 1968 NATO conference on the "software crisis," where it was recognized that software was routinely late, over budget, and simply wrong. To enumerate a list of such processes, paradigms, techniques, and methodologies here would be too arduous, but for the most part it, is generally accepted that the construction or engineering of software involves:

- Need
- Craftsman
- Compensation

In other words, some *individual or group in need* of software obtains that software product from a *programmer or group* for some amount of *compensation*. This is nothing more, really, than the law of supply and demand, which has been tested throughout human civilization. Following such law, if there is no "need," then there is no one to compensate the craftsman for their product, and hence no product. As such, nearly all defined processes for software engineering include some role for the end-user or defining-user in a software engineering process (such as requirements engineering (IEEE 1990) or "use cases" and "actors" in the Rational Unified Process (Jacobson, Booch, and Rumbaugh 1999)). Further, software engineering is concerned with the principles behind effectively organizing the team of engineers that craft the software, and also with how that craftsmanship is accomplished, in relation to:

- Designing the software (its architecture, modules, and interactions)
- Programming, or coding, the designed software
- Testing the software against design and need
- Documenting that which is designed, programmed, and tested
- Managing those that design, program, test, and document

Through the short history of rationalizing the process by which software engineering is, or should be, accomplished, the members of the SE community have reached a fairly common understanding of what software engineering is, and how software engineering should be done. It is the apparent departure of free and open source software (F/OSS) from this understanding (or belief in that understanding), combined with the success (or perceived success) of many F/OSS projects, that has attracted the attention of many in the research community. In this section, a number of authors have been selected to bring to the foreground specific observations from various F/OSS projects.

Mockus, Fielding, and Herbsleb (chapter 10) embarked on an empirical study by examining data from two major F/OSS projects (the Apache HTTP server and Mozilla) to investigate the capacity of F/OSS development practices to compete and/or displace traditional commercial development methods.[8] German (chapter 11) proposes that the actual design of the software (its architecture) is one organizing principle behind the success of the GNOME project, in that it supports open and distributed software engineering practices to be employed by a large number of geographically dispersed code contributors. German then corroborates those practices with empirical evidence from the records available for the project to measure the efficacy of those practices.

Following this, Jørgensen (chapter 12) traces the development cycle of releases of the FreeBSD operating system and presents the results of a survey of FreeBSD software developers, which was conducted to understand the advantages and disadvantages of the software engineering practices used by FreeBSD. The conclusions gleaned from his observations interestingly suggest that a strong project leader is *not* necessarily needed for a F/OSS project (such as FreeBSD) to be successful, although he proposes instead that a well-defined software engineering process is, perhaps, critical.

Finally, Robbins (chapter 13) looks at the common practices used in F/OSS software development projects and at the tools available to support many aspects of the software engineering process. He also points out where the F/OSS community is lacking in tool support for other software engineering processes.

Part IV: Free/Open Source Software Economic and Business Models

Previously, we noted that F/OSS seems to challenge many accepted software engineering norms. It also appears to depart wildly from established

software business models, and indeed F/OSS companies, and hybrid proprietary-F/OSS companies have had to create new value offers predicated on software as a service, value of software use rather than value of software purchase, and so on. In this part, we present four chapters examining these new models and the changing relationships between customers and companies, and between companies and competitors.

In chapter 14, von Hippel argues that F/OSS offers extraordinary examples of the power of user innovation, independent of any manufacturing firm. He contends that markets characterized by user innovation "have a great advantage over the manufacturer-centered innovation development systems that have been the mainstay of commerce for hundreds of years" and discusses, convincingly, the parallels between the F/OSS communities and sporting communities also characterized by user innovation.

Krishnamurthy (chapter 15) discusses a series of business models that have emerged in relationship to F/OSS. He articulates the relationships that exist between producers, distributors, third parties, and consumers, and examines the impact of different licensing structures on these relationships. In chapter 16, Dalle and David present a simulation structure used to describe the decentralized, microlevel decisions that allocate programming resources both within and among F/OSS projects. They comment on the impact of reputation and community norms, and on the economic rationale for "early release" policies.

Finally, Matusow (chapter 17) presents the perspective of what has always been the archetypal proprietary software company in the eye of the F/OSS community; namely, Microsoft. In discussing the Shared Source program and related initiatives, this chapter provides interesting insights into the impact that F/OSS has had on the proprietary software industry and, perhaps, vice versa.

Part V: Law, Community, and Society

It has been said that the average Navajo Indian family in 1950s America consisted of a father, mother, two children, and three anthropologists. Many in the F/OSS community no doubt are starting to feel the same way, as what began as a software topic has attracted the efforts of so many researchers from sociology, economics, management, psychology, public policy, law, and many others. The final section of the book presents research focused on legal, cultural and social issues.

Lessig (chapter 18[9]) paints a broad picture and challenges us to think about the social implications of F/OSS and the drivers behind the

phenomenon. Starting with the collapse of the Berlin Wall, he considers the move from closed to open societies. He discusses the U.S. model, where the move to having more property that is "perfectly protected" is equated with progress. For Lessig, the issue is not whether the F/OSS development model produces more reliable and efficient software; rather it is about the future of an open society drawing on F/OSS principles. Lessig focuses on the enabling power of combining a "commons" phenomenon with the concept of "property" to stimulate creativity, and also the critical differences between ideas and "real" things. Lessig also identifies the specific threats to ideas posed in cyberspace, a space that is not inherently and perpetually free but can be captured and controlled. Lessig offers a number of compelling examples of double standards where large corporate U.S. interests use the power of copyright law to prevent free communication of ideas, whereas they would presumably decry such curtailments on free communication if they occurred in other parts of the world.

As Niels Bohr once remarked about quantum physics, if it doesn't make you dizzy, then you don't understand it, and the same may hold true for F/OSS licenses. McGowan (chapter 19) deconstructs the legal issues surrounding F/OSS licensing. He presents a primer the structure of F/OSS licenses ("how they are designed to work") and a discussion on copyright, "copyleft," contract law, and other issues that affect the enforceability of licenses ("whether the licenses actually will work this way if tested"). His discussion of the Cyber Patrol hack and the Duke Nukem examples make these complex issues very concrete and accessible.

Moving from licensing to liability, O'Mahony (chapter 20) addresses the fact that as F/OSS moves more into the mainstream, the incorporation of projects as a mechanism to dilute the threat of individual legal liability becomes central. However, incorporation brings its own set of problems, in that it imposes a degree of bureaucracy that is anathema to the hacker spirit of F/OSS. O'Mahony deals directly with this conflict, an issue exacerbated by the many F/OSS developers operating on a voluntary basis with nonstandard systems of rewards and sanctions. O'Mahony identifies a number of dilemmas that have emerged as F/OSS has become more popular and the original hacker ethos and values diluted. She discusses the different incorporation models that have emerged historically and considers why they are inappropriate as organizational models for F/OSS projects. She then compares the foundations created by the Debian, Apache, GNOME, and the Linux Standards Base projects to study how different project "ecologies" approached the task of building a foundation at different points in time.

In chapter 21, Kelty elaborates on the oft-noted parallels between F/OSS and the scientific enterprise. He considers the extent to which they are similar, and also the extent to which F/OSS has (or will) become a necessary enabler of science. He focuses in particular on the social constitution of science—the doing of science, the funding of science, and the valuing of science—and draws parallels between the norms, practices, and artifacts of science and F/OSS. The chapter also considers issues related to law, thus resonating with McGowan's chapter earlier in this section. Likewise, his consideration of the threats facing science (and thus society) are reminiscent of those identified by Lessig.

The chapter by Szczepanska, Bergquist, and Ljungberg (22) illustrates the manner in which researchers can apply an ethnographic perspective to the study of F/OSS. They characterize open source as a social movement, and trace its origins in the literature on the emergence of the network society. They situate F/OSS as a countermovement in opposition to the mainstream IT culture as exemplified by companies such as IBM and Microsoft. Thus, their analysis resonates with the motivation factors identified earlier in the book. The authors use discourse analysis to analyze how the OSS community is molded and to help understand how collective identity is created and communicated. Understanding these discursive practices is especially important because of the decentralized and networked character of the OSS movement. The construction of the hacker is discussed, and the tensions between the Free Software and Open Source movements are analyzed. They further analyze "us" versus "them" constructions in the discourse of the community, and the discursive strategies of the anti-OSS constituencies. Interestingly, the rhetoric of the "American Way" is used by both pro- and anti-F/OSS communities to support their arguments. Finally, the authors consider the power relationships implied by a gift culture, and how these structure the work patterns of the F/OSS community.

Aigrain (chapter 23) who has written much on F/OSS, has drawn on his many years of experience with the European Commission to analyze their policy in relation to F/OSS. (He uses the term *libre software*.) The F/OSS phenomenon is arguably better supported by public bodies in Europe than in the United States, and European Commission support for F/OSS represents a very significant factor in the future success of F/OSS initiatives. Aigrain's analysis identifies choke-points in the EU funding bureaucracy that will deter many F/OSS practitioners, as well as important policy issues of which potential F/OSS researchers in Europe need to be cognizant. Aigrain suggests that, until recently, there was limited awareness of F/OSS issues

in the Commission, but that the growing disenchantment with the dissemination and exploitation of software research funded under the traditional proprietary closed model was an important motivating factor. He also identifies as an important motivator the desire to establish an information society based on the open creation and exchange of information and knowledge. Other drivers include concerns about security, privacy, and overreliance on a small number of monopoly suppliers of proprietary software. Aigrain also acknowledges the prompting by advocacy groups such as the Free Software Foundation Europe (FSFE). He insightfully notes need for sensitive support for the F/OSS hacker community in managing the statutory reporting requirements of a funding agency such as the EU. Despite this pragmatism, over the 1999–2002 period, only seven F/OSS projects were approved, with a total budget of €5 million, representing only 0.16 percent of the overall EU IST program funding for research. Aigrain also identifies some challenges for libre software, specifically in the areas of physical computing and network infrastructure, the logical software layer, and information and contents layer.

Finally, in chapter 24, O'Reilly presents a thoughtful and informed essay on F/OSS "as an expression of three deep, long-term trends"; namely, the "commoditization of software," "network-enabled collaboration," and "software customizability (software as a service)." He argues that it is by examining next-generation applications (the killer apps of the Internet, like Google) that "we can begin to understand the true long-term significance of the open source paradigm shift." More to the point, O'Reilly asserts that if we are to benefit from "the revolution," our understanding must penetrate the "foreground elements of the free and open source movements" and instead focus on its causes and consequences.

Rigor and Relevance

We believe that academic research should be both scientifically rigorous and also highly relevant to real-life concerns. We also believe that good research answers questions, but great research creates new questions. Thus we conclude this introduction with some suggested questions for you to keep in mind as you read the book. We've grouped the question into three audience-specific lists for F/OSS project leaders and developers, managers and business professionals, and researchers and analysts. We suspect most of our readers, like most of our authors, wear more than one of these hats.

F/OSS Project Leaders and Developers

- What are the major motivations for the developers in your project?
- Is your project culture such that it can accommodate developers with different motivations to participate? Or does your project risk crowding out developers by having a culture that supports only a single motivation to participate?
- How can you manage both paid and volunteer contributors?
- On what basis do you welcome new members and how can you integrate them into your community?
- How can you best manage the "economy of talent" within your project? How can you settle disagreements and disputes? How can you avoid (destructive) churn?
- How can you manage software complexity? Integration? Testing?
- How can you break the "security symmetry" created by F/OSS?
- How are communication and collaboration facilitated in your project?
- How are changes from the F/OSS community accommodated?
- Can you automate day-to-day activities? What tools do you need to use?
- How can you leverage user innovation? How do you enable your users to contribute to the project?
- Is your project part of a commercial business model/value web? Where does your project fit in?

Managers and Business Professionals

- How can nonfinancial incentives be utilized within your firm's software projects to motivate internal developers?
- How can you spark the essence of creativity among your software developers?
- How do you build an open community of sharing and peer review within your firm?
- How does your firm interact with the wider F/OSS community? What things do you need to be aware of so that you do not drive out F/OSS developers?
- How do you leverage the increasing numbers of F/OSS developers for the benefit of your firm?
- What criteria are important in your evaluation of F/OSS products? How does your procurement process need to change to adjust to F/OSS?
- How do your implementation and change management processes need to change to adjust to F/OSS?

- In what way do your existing processes (or tools) have to adapt to support F/OSS development?
- What criteria do you need to choose a F/OSS license? Or, if you are attempting to emulate the F/OSS process without using F/OSS licensing structures, what challenges do you anticipate?
- What can your firm learn about collaboration and agility from F/OSS project organizations? What can they learn from you? (Remember, you can contribute knowledge, not just code, to the F/OSS community.)
- What business model(s) is your firm engaged in? What role do F/OSS products play in your value offer? F/OSS processes? F/OSS communities?
- How can F/OSS play a role in your firm's "corporate citizenship"?

Researchers and Analysts

- Does the F/OSS phenomenon shed new light on how creativity works in knowledge workers?
- What is it about programming that evokes a creativity response in software developers? Can this be achieved in nonsoftware environments?
- What are noneconomic incentives to innovate in complex product industries?
- How portable are F/OSS motivations and practices to other domains of economic activity and social organizations?
- How can F/OSS processes be utilized in proprietary settings, and vice versa?
- How can F/OSS tools be utilized in proprietary settings, and vice versa?
- What are the weakness of the F/OSS process and toolkit? How can these be addressed?
- What are the strengths of the F/OSS process and toolkit? How can these be leveraged?
- Do the dynamics of F/OSS create new opportunities for research (new methods for data gathering and analysis)? If so, what are the ethics involved?
- Does the F/OSS phenomenon force us to rethink the nature of innovation?
- Does the F/OSS phenomenon force us to rethink the nature of work?
- Does the F/OSS phenomenon force us to rethink the nature of knowledge sharing? Of intangible/intellectual assets?
- Is F/OSS overly reliant on a countercultural identity? How does "success" change the F/OSS process?
- What are the relationships between F/OSS and other forms of creativity and knowledge creation?

- Does F/OSS provide new modes of organizing and collaborating? What are they?
- How does F/OSS actually help address the "digital divide" and the needs of the information society?

Notes

1. http://www.gnu.org/philosophy/free-sw.html.

2. http://www.opensource.org/docs/definition.php.

3. See Feller and Fitzgerald (2002) for a fuller discussion of this. Several of the chapters in this book also address the issue, directly or indirectly.

4. You'll find all three terms (and every possible combination) used by the various authors who wrote the chapters in this book—we let people choose their own labels, rather than normalizing the book with unintentional side effects.

5. Most of the publicly available references in the bibliography of this book can be found in multiple citation management formats (EndNote, Bibtex, and so on) at http://opensource.ucc.ie. Additionally, full-text versions of many of the papers cited are also available in the research repository at http://opensource.mit.edu. We hope that these two resources will be very valuable to our readers.

6. Fear, Uncertainty, and Doubt.

7. Other definitions of software engineering include these same concepts, but go on to include economic aspects (for example, "on time" and "on budget") as well as team management aspects (SEI 2003).

8. Chapter 10 is an edited reprint of Mockus, A., Fielding, R., and Herbsleb, J.D. (2002), "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, 11:3, pp. 309–346.

9. The contents of chapter 18 were originally presented by Lawrence Lessig as a keynote address on "Free Software—a Model for Society?" on June 1, 2000, in Tutzing, Germany.