Abstract: 3
I. Introduction: 4
Chapter I: Overview of the Open Source Movement 8
A. Chapter Overview: 8
B. The Origin and Future of Open Source 9
1. The Rise of Open Source 9
2. Open Source Innovation 15
3. The Business of Open Source 18
C. Demographics: Who Participates in Open Source Groups and Why? 24
D. Chapter Conclusions: 27
Chapter II: Theory and Literature 28
A. Chapter Overview: 28
B. New Work Organizations in Software Development 29
1. Role and Task Negotiation in Non-Hierarchical Organizations 29
2. Control and Commitment: The Ideology of Open Source 41
C. Administrative Control and Software Development 44
1. Managerial Attempts to Control Programmers 44
2. An Alternative Model of Professionalism: The Occupational Community 50
D. Chapter Conclusions 57
Chapter Three: Methodology 60
A. Chapter Overview 60
B. Research Agenda 61
1. Research Questions and Brief Summary of Results 61
C. Methods and Data 64
1. Methods 64
2. Sampling Design and Population 65
Table 1: CHARACTERISTICS OF PRELIMINARY INTERVIEWEES 66
3. Data Collection 68
4. Validity and Reliability 69
5. Data Analysis 72
D. Chapter Conclusions 74
Chapter IV: History, Culture and Stages of Group Growth **Error! Bookmark not defined.**
Introduction: **Error! Bookmark not defined.**
A. Common Threads of Belief **Error! Bookmark not defined.**
B. LithiumBSD: A Guild of UNIX Gurus **Error! Bookmark not defined.**
C. Star Linux: A 'Stepping Stone' for New Programmers: **Error! Bookmark not defined.**
Chapter V: Coordination and Task Assignment 86
Introduction 87
A. Bug Tracking and Coordination 89
DIAGRAM 1: FORMAL AND INFORMAL COORDINATION SYSTEMS 94
B. Creation of Group Practices and Policies 100
Chapter VI: Role Negotiation and Task Assignment 106
Introduction 106
B. Role Negotiation and Conflict Resolution 111
1) Initiation into Groups: 111
2) Role negotiation 115
3) Negotiating High Status Positions 124
4) Leadership 132
5) Conflict Resolution 138
DIAGRAM 3: COMMON TYPES OF CONFLICTS AND METHODS OF RESOLUTION 145
Chapter VII: The 'Open Source Movement' as an Evolving Professional Community 147
Introduction 147
A. Professional Motivations for Open Source Participation: Transparent Systems 151
B. Open Source as an Evolving Professional Community: Transparent Organizations[2] 154
C. The Professional Ideologies of Open Source Programmers 158
Chapter VIII: Coordination and Role Negotiating within Two Subprojects 165

A. The Star Linux Rules Committee: A 'Transparent' Project 165
1) Group History 166
2) Group Coordination and Tasks Assignment 167
3) Role Negotiation and Conflict Resolution 169
B. The LithiumBSD Foo Project: A 'Trailblazing' Effort 177
1) Group History 178
2) Group Coordination and Tasks Assignment 180
3) Role Negotiation and Conflict Resolution 182
Chapter IX: Implications and Conclusion 188
Introduction: 188
A. The Resurgence of the Craft of Programming 189
B. The Work Practices and Culture of Two Post-Bureaucratic Organizations 194
C. A General Model of Post-Bureaucratic Group Growth 196
Stage 1, "Trailblazing" (Estimated size: 5-25 members): 199
Stage 2, "Passing the Torch" (Estimated Size: 25 – 300 members) 202
Stage 3 "Transparent Organization" (Estimated size: 300-5000+ members): 206
DIAGRAM 5: CHARACTERISTICS AND STAGES OF GROWTH FOR GROUPS STUDIED 211
Appendices: 212
A. Principles of the Postburecractic Ideal Type, from Charles Heckschler, 1994. 212
B. Protocol for Interviews with Committers and Leaders 215
C. Brief Statement of Findings from Preliminary Interviews 218
1. Role and Task Negotiation 218
2. Open Source as an Occupational Community 220
C. Midpoint Analytical Note 221
1. Reasons for Participating in Open Source Projects 221
2. Project Management 223
3. Work Roles and Coordination 225
4. Hierarchies of Knowledge and Group Conflicts 232
5. Open Source as a Foundry for Professional Skills 235
Appendix D: Example Work Practices Described by Members 237
Appendix E. Glossary 238
Bibliography 240

Abstract:

Over the past twenty years, a rapidly growing community of programmers who volunteer, or more rarely are paid to develop 'free software,' including the Linux and BSD operating systems and the Apache web server, has grown exponentially and has developed software which is gaining increasing adoption by individuals, businesses, governments and other organizations. This dissertation is an ethnographic study of two open source programming groups conducted with two primary purposes in mind. The first objective is to understand the culture and work practices that allow post-bureaucratic organizations to hold together in the absence of well-defined work roles and responsibilities. Insights into these processes were provided through in-depth interviews with approximately twenty-five members of each group which lasted one to two hours each, as well as participant observation of group practices. Calvin Morrill's study "Conflict, Honor and Management Change," an ethnographic study of the culture and practices that evolved within a traditional bureaucratic firm which adopted the relatively non-hierarchical 'matrix' structure of management, was used as a theoretical and methodological model for this study. This dissertation also builds upon an 'ideal type' of the post-bureaucratic organization outlined by Charles Heckscher in *The Post-Bureaucratic Organization.* Heckscher provides twelve principles which post-bureaucratic organizations are expected to hold. These principles are cited in the Appendix and are quoted, when applicable, before the chapters that demonstrate how these principles function in practice.

The second major objective of this dissertation is to examine whether the 'open source movement' is an evolving occupational model for professionalizing software engineering

which students of professionalism have long labeled an 'imperfectly professionalized' occupation. In "Bureaucracy and Craft Administration of Production," a seminal article on different forms of bureaucracy in organizations, Arthur Stinchcombe argued that more complex or 'organic' labor such as programming is best managed under 'craft' divisions of labor where work practices are managed by workers themselves rather than the more bureaucratic 'administrative' division of labor, which is better suited for managing more 'mechanical' labor (Stinchcombe, 1959). The open source movement appears to be an emerging craft model of production for software development which attempts to solve many of the problems traditionally faced by organizations attempting to manage programmers including a high failure rate for software development (Gibbs, 1994). By demonstrating that they are able to create software products that perform as well or better than those developed by traditional software companies, open source programmers are making a compelling argument both for their organizational approach towards developing software outside of bureaucratic organizations and their legitimacy of programming as a self-policing profession.

I. Introduction:

Over the past decade, there has been an exponential increase in the number of computer programmers, professional and amateur, volunteering their time to participate in 'open source' groups, which develop a wide variety of free software[1]. Open source is a method of developing software by opening the source code (the instructions that tell the

program what to do) to the public, in contrast with traditional methods, which require that this information be protected as intellectual property. Open source groups vary in size from a few developers to hundreds of worldwide members, and these groups focus on different types of programs including operating systems, internet messaging clients, and many others, most notably the various Linux and BSD operating system projects. While most software corporations rely on a limited pool of paid contributors to develop software, the main advantage of open source development is that developers with different skills and interests can contribute to the project. Despite little or no funding, the lack of a formal structure, and reliance on volunteer labor, many open source groups have been able to create software, which equals or exceed the performance of proprietary solutions, and often within a shorter timeframe (Mockus, et. al., 2000; Wang and Wang, 2001).

Open source has also become increasingly important in the business world, a trend which is evident both in the increasing adoption of open source software by corporations (e.g. 60% of web servers are run using the open source Apache server) and by the increased involvement of software companies in open source projects. Most major software companies and many hardware companies, including HP, Apple, Sun, and many others, have sponsored open source ventures to tap the resources of skilled programmers willing to donate their time to a high profile project. Not all companies who make their living from selling software are willing to make their code freely available, and a few, including Microsoft, have attempted to discredit open source. The acrimoniousness of the debate between these competing philosophies was recently evident with the firing of a

[1] The Open Source Development Network, osdn.org, currently hosts about 70,000 projects in areas as diverse as Database Management, Security, Education and Sociology. See Chapter V, Section B for an

"senior strategist" at Hewlett-Packard who was a strong advocate for open source, apparently because the merger with Compaq would require HP to maintain a strong association with Microsoft (Lohr, 2002).

In this study, I will attempt to understand how open source groups are able to coordinate software development in the absence of formal management structures and incentive systems. I will consider whether the rise of the 'open source movement' presages the rise of a new occupational community for programmers, and the lessons that open source groups have for managing post-bureaucratic organizations. There have been relatively few ethnographic studies of post-bureaucratic organizations in practice (See Morrill, 1991 and Kunda, 1993 for notable exceptions) or of the process of role and task negotiation required by workers as bureaucratic structures break down due to new technologies and other pressures (See Morrill, 1991; Barley, 1990; Zetka, 2001; and see Freidson, 1963 and Strauss, et al, 1964 for foundational research on role negotiation). Finally, few empirical studies of the emergence of occupational and professional communities have been undertaken (See Van Maanen and Barley, 1984 for an overview of research in this area, and Perrucci and Gerstl, 1969 for an early study of the engineering community).

This study will consider two aspects of the open source phenomena: the negotiation of roles and task coordination within a non-hierarchical organization, and whether the 'open source movement' is a form of occupational resurgence for software developers. I will consider whether the open source movement is an evolving global 'occupational community' which provides a form of peer review for computer programming not typically available to programmers, and socializes members into occupational norms and

---

overview of the open source movement.

work roles.

Chapter I: Overview of the Open Source Movement

A. Chapter Overview:

Software developed by open source groups has gained increasing popularity on many fronts including endorsements by many local and national governments, and participation and funding by many corporations. Governments that formally endorse the use of open source software include Germany, France, the Philippines, Peru, Mexico and China, and in September of 2000, the White House published a report recommending the use of open source for "government's high-end computing needs" (Computers and Security, 2001).

Although profiting from open source can be difficult[2], open source solutions have become popular for reasons other than their low price. For example, in the entertainment industry open source has been used to digitally enhanced movies, which requires the use of cutting edge technology (Salon, 2001). Open source servers run with Apache software also provide the backbone of the Internet, and currently Apache has over 60% of the webserver market share (Mockus, et. al., 2000). Another area of growth for open source projects has been through investments of manpower and money from companies eager to associate themselves with the open source movement, including Oracle, Apple, Netscape, Sun and IBM[3]. Almost every major hardware and software company has participated in an open source project to some extent with the notable exception of Microsoft, which remains vigorously opposed to the "free software" movement. This chapter charts the rise

[2] See Chapter I, Section D
[3] See http://www.ibm.com/linux

of the open source movement and attempts to explain the reasons for the rising popularity of “free software.”[4]

B. The Origin and Future of Open Source

1. The Rise of Open Source

Open source is a ground-up method of software development, which according to past methods of doing business simply should not work. Briefly, in this model of development the source code of programs (the instructions that tell a program what to do) becomes public domain. Any user with the technological know-how can modify the software to suit their needs; the only requirement is that they make their modifications available to the community.

There is no formal assignment of tasks; rather users choose to work on projects as it suits them, according to their skills and intellectual interests. Open source programmers volunteer their time to solve problems that interest them and can abandon projects as it suits them, shifting responsibility to other interested programmers. This decentered model of development divides tasks in a somewhat egalitarian fashion: any member of the group can work on any aspect of the project, and they will be rewarded by having their contribution added to the source code, if they are able to convince the rest of the group that their contribution is worthwhile. Projects often live beyond the contributions of their creators, because if the "owner" of a project is no longer interested in the program, she can pass it on to someone else.

[4] This chapter does not provide a full history of the open source movement. A more comprehensive history is available in *Free for All* (Wayner, 2000) and *Hackers: Heroes of the Computer Revolution* (Levy, 2001).

The quality of a programming innovation is determined through a type of "peer review": other technically sophisticated members of an open source group examine proposed additions to the program and decide whether the change improves upon current code. Discussion lists where users share their opinion with the group are a key resource for this type of decisionmaking. Anyone can post criticisms of the contributions of others as well as suggestions for areas in the code that need improvement. There is rarely a formal decisionmaking process, rather discussion continues until there is a group consensus on which is the "best" solution is for a software problem. This collective model of decisionmaking ensures that group efforts are only directed toward the problems that are 'collective annoyances' for the group and since the group is (at least initially) the pool of users for a new product, this ensures that 'product development' closely follows 'customer' needs. If a member wants an innovation in which there is little interest, she can always program the 'hack' herself.

The original 'kernel' for the Linux operating systems, one of the most visible open source projects, was developed by Linus Torvalds, a Norwegian computing student (one of only two at his University) who developed the system as a hobby, when he found that buying an operating systems that had more than basic functionality was too expensive. He essentially built the system from scratch--there were no accurate explanations of the inner workings of his computer or its operating system so he spent many months learning how to write assembly language instructions (the native digital language of the computer) to write text to the monitor. After several months he was sufficiently skilled to write a program that allowed him to move a pixel across the screen--this was the first step towards writing text. This project was a complicated effort and did not promise any

immediate economic or academic rewards, but Torvalds' interest in the project persevered. As the system grew to allow access to the hard disk and I/O ports, Torvalds posted his code to an early Bulletin Board Service.

Interested users could download his innovations and criticize mistakes, posting suggestions and 'hacks' of his program back to the board. Much of his efforts in creating a user interface were build around software created by an early pioneer of the open source movement, Richard Stallman. Stallman developed the GNU interface in opposition to Berkeley's management of Unix, which he saw as dependent on corporate and military interests (the recursive acronym stands for Gnu is Not Unix). To gain independence from outside interests, Stallman rebuilt much of the Unix system from the ground up. This legendary hacker provided two fundamental pieces to the Linux puzzle: the compiler which allowed Torvalds to build the kernel for the Linux system and the GPL license, or the "copyleft" (a pun on copyright) which specifies that users, corporations etc. can make use of the program and modify it as they please as long as their modifications are released back into the public domain.

During several stages of the development process Torvalds considered scrapping the project and only when he found new problems that interested him did he continue to program. Gradually interest in his project increased as other techies who were seeking operating systems encouraged his efforts and then began to contribute their own elaborations on his kernel. He accepted new solutions if they improved the code and decided between competing submissions or "patches" based on group agreement regarding which submission was the technically best solution. His operating system has evolved into a product, which is far more stable than Windows (less system crashes) with

some users reported using their system for years without having to reboot it. Other open source software has become increasingly accepted as well: Unix systems currently run a large portion of academic and internet networks and Apache systems (an open source operating system for servers) are used in the majority of servers today.

Torvalds believes that one day his operating system will challenge Microsoft for the desktop market. Linux systems are becoming the standard for corporate servers and they have also been embraced by many governments, including China as well as several government agencies in the United States, partly because government agencies are often plagued by both incompatible computer systems and a limited Information Technology (IT) budget. However, Linux has only started to gain acceptance in the desktop market.

The nature of the open source model explains why companies are rapidly adopting open source software and why typical home users are less likely to use it. In part, the growing popularity of open source software is understandable because the price of much of this software is very low or free, reducing software overhead, but Linux systems are also considered by many to be superior technologically as well. The server market was the first to adopt the open source code because its users are more technically literate and more willing to forgo easy installations and a lack of help manuals, although there is a great deal of help available through informal user groups. In contrast, less technically sophisticated users are typically satisfied with easy to use, if less customizable, and often less reliable solutions.

The diverse array of Linux modifications made by users with different systems and problems also aids in its "portability" or ability to function on many different types of computer systems. For example, while there are interoperable versions of Linux for

systems as diverse as the Mac and Atari, Microsoft has focused only on the PC market. In contrast, the "cookie-cutter" approach used by Microsoft and other proprietary software developers attempts to create one version of their software that will work on the systems that are most popular.

It is easy to understand why Microsoft is threatened by this model. A Senior Vice President, Craig Mundie, recently attended an Open Source conference, arguing that open source development does not work as a business model and emphasizing "the importance of intellectual property rights"[5] in protecting the software market. The traditional model of software development cannot compete with companies that almost give away their product, while allowing competitors access to their innovations.

One major effort by open source programmers, now that a graphical user interface has been developed and most of the bugs ironed out, is creating software that is compatible with Windows applications. Currently, users have developed their own Linux-specific versions of applications including word processors, image editors, etc., which may not be able to modify files created with Windows-based systems, limiting their adoption by most business and home users. Torvalds emphasizes that Microsoft is not “the Enemy” but “eventually” he expects Linux to steal the desktop market from Microsoft as well as the server market. In addition, some corporations are fielding their own open source efforts, including IBM. In an article titled "IBM is ready for Linux. Are you?" the Vice President for Linux Marketing and Sales describes the large investment IBM is making in both creating their own versions of Linux software and in marketing the Linux brand. He explains that Linux "breaks new ground in terms of flexibility, portability and

[5] www.microsoft.com/presspass/exec/craig/05-03sharedsource.asp

interoperability" creating "an operating system that's uniquely Web-ready" because Linux can be adapted to fit many software and hardware needs.

This contrasts with attempts by Microsoft and others to create one set of universal "standards" or Cathedral, which fulfills all user needs. Some software and hardware companies have begun similar efforts to make Linux friendly versions of much of their software. Other corporations have attempted to "channel the magic" of the open source development model in improving their own products, with varying success. For example, Netscape released the source code to its Mozilla browser to allow of the programmers to create a web browser that will compete with Microsoft's Internet Explorer.

There are two primary barriers that have prevented the Linux and other open source operating systems such as BSD from gaining acceptance among the public for the desktop market. The first requirement for mass acceptance of open source projects is that the application be easy to use, install and fix problems with. The rise of corporations which profit from open source, such as RedHat Linux, have eliminated many problems such as improper documentation, difficult installations, and lack of technical help. The second barrier is that Linux must work with existing applications--almost overwhelmingly Windows applications and open source efforts such as the Wine emulator and Samba are progressing towards the goal of allowing Windows software to run within open source systems. The grass-roots origin of open source groups outlined above has let to the evolution of a model of software development, which is relatively insulated from the pressures of the business world and academia, and which may have lessons for traditional model of managing software projects.

2. Open Source Innovation

In its purest form, the open source model of development is chaotic: users often develop competing solutions to a problem, apparently wasting group effort since only one version can be incorporated into the kernel. But there is some order underlying the chaos of open source development. These groups are typically governed either democratically, where every full member of the group (or “committer”) is allowed to decide which direction the project will take. A second model of governance is the “benevolent dictator” model, where the initiator of the project allows others to improve upon the kernel, but retains final say in the most important decisions. Finally, some projects have established elected bodies that help to coordinate work tasks and to resolve conflicts. These systems of organization prevent radically competing efforts within the same project, but if radically different views remain within a group, users can "fork"--that is start their own competing project. This is the process that led to the many versions of Linux and other open source programs currently available. One of these versions or ‘distributions’ is the subject of this study which will be referred to as ‘Star Linux’ as well as ‘LithiumBSD,’ one ‘fork’ of the BSD operating system.

This diversity in the many versions of the operating system is one of the strengths of Linux and other open source projects, however, for two main reasons. First, the variety of competing versions of Linux encourages specialization, while ultimately retaining the same core kernel. Secondly, this diversity of efforts has led to versions of Linux that work on many different "platforms" (e.g. Macs, many different types of servers, Sega DreamCast, and typical home PCs). Programmers who wanted Linux software for one of

these systems simply modified it to fit their needs. As a result the code for Linux is very flexible, the very antithesis of proprietary software systems that require users to buy upgrades and services only from the original corporation and which are optimized for a single hardware configuration. This diversity of versions of Linux, also know as "portability," is a selling point for corporations who often are running many different operating systems with different hardware configurations. Proprietary software, on the other hand, allows little or no modification of its source code without the special permission of the owner of the software.

The culture of these groups appears to act as an organizing force within projects. Many observers of the 'open source movement' have found that communities are held together by a shared commitment to the 'hacker ethic'. According to Steven Levy, one pioneer of the computing industry and the open source movement, the 'hacker ethic' emphasizes that only by getting into the 'guts' of the system could hackers understand how technologies work, allowing hackers to improve on them[6]. Hackers distrust 'user friendly interfaces' and one's standing in the hacker community is based on the ability to find the 'best' solution to a technical problem, often through innovative and unexpected uses of other technologies or clever programming. Ideally, the quality of a 'hack' should be visible to anyone with the technological know-how to appreciate its elegance.

Communities of technical specialists have long historical roots and can be traced back to craft guilds. As users of mechanical, electrical, and other systems began to experiment with innovative and pragmatic solutions, they sparked new inventions and industries. These innovations came "from the ground up" as those who knew the machines and

[6] The term ‘hacker’ has only recently become associated in the media with destructive uses of technology and in the past has been used to refer a craftsman who “hacks” furniture, among other definitions.

technologies the best "played around with" or "hacked" the systems to create better systems. The revolution of the open source model comes in part from the possibilities afforded by electronic communications as well as the nature of their product itself, which unlike tangible machines, could easily pass from user to user around the world. The open source community has grown rapidly because electronic forms of communications, such as email discussion lists, allow for rapid formations of “communities of interest.” Similarly, the "guts of the machine" are easily accessible to anyone with internet access who is interested in viewing the source code of an open source project.

It has been noted that both the 'peer review' aspect of open source programming and the creation of a pool of shared knowledge parallels scientific communities and other professions which distinguish themselves by their ability to expand the existing body of occupational knowledge, in accord with the norms of scientific openness. When this common stock of knowledge is converted into private property by restricting it only to those who are 'qualified', according to open sourcers, innovation is stifled.

One 'leader' of the open source movement, Eric Raymond, argues that the strength of the open source model can be understood as a comparison between two systems of organizing group efforts: 'the Cathedral' and 'the Bazaar'. While one Architect (e.g. Bill Gates) organizes the Cathedral and has control over the whole organization, many different sellers with competing “products” or ideas fill the Bazaar. Raymond argues that, over time, attempts to organize workers within a single well-defined bureaucratic system will fail to create the best software, while under the Bazaar model, "many different Linux versions [will] proliferate, but over time the marketplace of software [will] coalesce around the best standard version" (Raymond, 2002).

In order to encourage the participation of many competing “sellers” in open source development, rather than protecting the purity of his doxa by not sharing his source code, Linus Torvalds followed the model of: "release early and often, delegate everything you can, and be open to the point of promiscuity" (Torvalds, 2001). This ensures that there will be no barriers preventing interested programmers from 'hacking' his code to create innovations.

While critics of the bazaar/cathedral comparison point out that at the end of the day there is only one individual who makes the final decisions relating to an open source project, the 'benevolent dictator’, if users are not satisfied with the direction that a current project is taking, they can always take the source code and create their own version. The diversity in versions of Linux and other open source software also aids innovation because any innovation is ultimately made available to all competitors, who can then incorporate it into their source code. As a result, the main division between competing versions is that they solve different problems (e.g. working on a Mac or creating a highly secure operating system) or that they include unique proprietary software, or services such as technical help. Under the open source model, however, it is impossible to compete on the strength of your product alone because the software itself if free.

3. The Business of Open Source

As open source projects became more successful, many companies and individuals have attempted to profit from “Free software” by using non-traditional business models.

Companies, such as RedHat, that make money from selling open source software are banking on the fact that many users prefer prepackaged installation with a user manual and technical support for a fee, to downloading and assembling the files themselves for no cost. This creates a "hierarchy of convenience" where typically only the most technically adept users have the know-how required to install and maintain open source software without outside help. Less proficient users pay a fee to buy a packaged and well-documented version of the software, or as the RedHat[7] corporate site states, "Open Source Software is free if your time is worth nothing." A diverse array of business models have developed in attempts to profit from open source software, including asking for voluntary contributions (often known as "freeware" or "shareware") or offering additional functionality within a free product for an additional fee.

The two most notable success stories are the RedHat corporation, the largest and most profitable open source company, and the Apache Foundation, which successfully licenses "free" operating systems to businesses for thousands of dollars by offering service and support. In an interview the CEO of RedHat explained that when potential customers complain about the price of the software he sells, he directs them to a company which sells his product on CD ROM for only $3 a copy--a CD which includes the entire operating system without documentation. Although it may appear daunting for a company to profit from selling open source software, alternate models have developed to make profits from providing services to users such as technical help and user manuals. In Free For All, Peter Wayner uses the Kool-Aid analogy: while customers of a supermarket can make Kool-Aid from the raw ingredients themselves, many opt to buy soda, which is

[7] A major distributor of Linux which gains profit by selling copies of their own *distribution* of Linux and by providing technical help and other services to users. The corporation is listed on the New York Stock

already prepackaged, and costs more (Wayner, 1999). Selling nicely packaged versions of RedHat software and other open source software which include technical support and documentation, have gained RedHat and other open source corporations a growing portion of the operating systems market. According to recent estimates, Linux is found on over seventeen million computers and RedHat has gained a respectable profit from their efforts.

According to traditional models of innovation and development, this business strategy should not work. In the traditional business plan for software development, “the best software is made by the best team that money can buy” and programmers work in secrecy, attempting to create more useful applications than their competition and to lock in a faithful pool of customers. While this business model has clearly been profitable for Microsoft and other software companies, critics argue that closed source software development encourages the release of buggy code in order to gain initial market share and customer dependence. Rather than hide their bugs in proprietary code, open source projects typically publish them and welcome other users to fix them and to adopt the product to suit their needs. In contrast, traditional corporations wait until their next version of their product is available to fix bugs, or attempt to create multiple "patches" which are often not well publicized. Traditional corporations take great pains to protect their “intellectual capital” because their processes of develop and ideas are seen as the foundation of the enterprise. Another important difference between these two business models is that traditional companies develop their software to fit the projected market trends and preferences of potential customers and they tend to create one product (e.g. Windows 98), which fulfills the need of the majority of users. This "widget" model of

---

Exchange.

development is an attempt to create applications that work for a wide variety of users possible.

This approach creates many "niche markets" for operating systems and applications for users who are not satisfied with the lowest common denominator. The first, but not only, niche that the open source movement worked to fill was that of technically sophisticated users who wanted access to the "guts" of the machine, its operating code. Despite this non-traditional approach to the business of software development, it appears that open source will become very profitable in the future.

One of the most popular markets for open source software, high-end computing needs for businesses has been growing steadily and according to one financial projection sales of Linux platforms will “triple to $6.5 billion in 2006” in comparison with a projected $27.7 billion in sales for proprietary systems (Fordahl, 2002).

Despite its many advantages to traditional models of software development, open source is not a panacea. Many corporations have attempted to adopt the open source model of software development, usually to a more limited degree. For example, while Netscape allowed users to modify the code to it’s web browser, the Mozilla project was not truly open source because the corporate "leaders" were not on par with the users: they retained the right to accept or deny every change made by committers. In addition, Netscape retained the copyright to the program, so that competitors are not able to use their code. While it is understandable for companies that lead open source efforts to want to lead development efforts and to retain control over the final product, they will not have access to the full strength of “distributed programming” and wholehearted programming contributions by volunteers, to the degree to which they retain the traditional model of

control. This bodes well for ground-up open source corporations such as RedHat and Debian, as long as they can find a way to make a profit from their efforts while remaining true to the ‘hacker ethic’.

It has been argued that many of these corporate open source projects have had limited success because they did not fully adopt the "hacker ethic": they could not let go of the corporate need for control over development and their intellectual product. Programmers will only form a "virtual community" around a development project if they feel they are contributing to a worthwhile project--simply working as the unpaid labor force for a major corporation will draw few of the best programmers. Rather, a community forms only if users have a stake in the project, either emotional or reputational, and if they are involved in decisionmaking in some way. For good or for ill a common dislike of Microsoft has fueled some projects as users work to topple the monolith. Other users contribute to a project with hopes that their efforts will be noticed by a corporation, who will seek them out or that their project will look good on a resume. This is not as unrealistic a hope as it may sound: source code of high profile open source projects is released with a list of all major contributors to the project. A final important difference between the “cathedral” and “bazaar” model of developing software is in the motivation of programmers. Within traditional software companies, programmers have their goals set by the CEO’s vision of the future of the market.

Programmers, it is assumed, are motivated by money and it is assumed that a high salary and other perks will draw the best talent. In return for good pay, programmers are expected to focus their efforts on projects and problems that management identifies as important. Typical models of business and management simply do not work when open

source corporations are formed. To successfully germinate open source development, management must use tools of persuasion to retain the best programmers. As Linus Torvalds argues, money is no longer enough of a motivation for the best programmers--they can pick and choose which efforts they want to join and often will take a pay cut if the project is interesting.

For all the strengths of open source development models, including access to a wide pool of programmers through a decentered network, and flexibility to the needs of its users, critics have identified several weaknesses of the model. First, the best programmers focus their efforts on the projects that interest them the most and not those which most need to be done. The more mundane tasks that would be solved in a traditional system may be ignored for more interesting tasks such as developing games or solving a problem that is interesting or useful for the programmer rather than important for the user (such as writing detailed user manuals).

Similarly, open source projects will not always meet the needs of specialized user groups. For example, if users want software that keeps track of medical records and billing, it will not "evolve naturally" under the pure open source model. Complex new fields (such as voice recognition or translation software) may be more difficult to organize ‘from the ground up’ because there may be no initial ‘kernel’ to which everyone can contribute until one ‘pioneer’ develops it from scratch, as was the case with Linus Torvalds. Incorporated open source projects such as RedHat, on the other hand, can focus on developing versions of these specialized applications in Linux with the hope of gaining customers and allowing current applications to work with Linux operating systems is the quickest solutions.

Another obstacle to mass acceptance of open source software is that most users of Linux, and other "hacked" programs are technically sophisticated--they do not have a problem downloading and compiling multiple files and enjoy reprogramming the code to fit their need. The average user balks at difficult installations, as well as a lack of product manuals and technical support. However these difficulties have begun to wane as corporations such as RedHat gain profit from supporting mainstream users, by putting all necessary files on one easy-to-use CD, for example. Finally, complex new fields (such a voice recognition or translation software) may be more difficult to organize 'from the ground up' because there may be no initial 'kernel' to which everyone can contribute until one 'pioneer' develops it from scratch, as was the case with Linus Torvalds.

C. Demographics: Who Participates in Open Source Groups and Why?

Open source groups can be divided into three major types. A small number of projects are established and highly visible within the open source community and draw a large number of participants. The most notable “high visibility” groups are open source operating systems including the various “distributions” of Linux and BSD and webservers such as Apache, and they draw a large base of users and participants. The second type of project involves a small group of programmers or a single programmer who wants to produce a less ambitious software package (such as a text editor, which duplicates the basic functions of Microsoft Word). These “low visibility” groups are typically hosted through the OSDN (Open Source Development Network) which includes

the websites SourceForce.net and Freshmeat.net[8]. Currently there are about 73,000 projects listed at OSDN (although some are projects which were "orphaned" when the project leader lost interest) and 610,000 registered users. Only a small percentage of these projects attain high visibility within the open source community.

Finally, hundreds of open source projects are initiated by hardware and software companies eager to enlist the aid of the open source community including OpenOffice, a suite of office tools created by Sun, provide much of the functionality of Microsoft Office. There is not a great deal of information available about participants in open source groups. One notable exception is the Boston Consulting Group's recent "Hacker Survey" which surveyed thousands of contributors to open source projects hosted at SourceForge to determine who they are and what motivates their participation (Lakhani and Bates, 2002).

According to this survey, participants in open source projects are overwhelmingly male (98% of sample), and about half reside in "the Americas" including Canada and Mexico. The remainder are European (42%) or from other countries. The typical age of respondents was 28, although the range is fairly wide, varying from 13 to 57 years of age. Finally, respondents tend to be IT (Information Technology) professionals and the majority are currently employed as programmers or in other IT positions (58%). The remainder are students (20%), "academics" (7%) and are in "other occupations" (15%). On average, respondents had 11 years of professional experience.

The high proportion of male participants is striking, especially in comparison with the gender representation in computer science and the computer programming profession.

[8] OSDN offers programmers resources such as free storage space and software to easily track communication between developers as well as bug reports, etc.

Estimates of the percentage of women pursing computing science degrees vary from 16% to 25% (Randal and Vardi, 2002; Margolis and Fisher, 2001) and women make up an even larger percentage of the computer programming profession. According to a recent report, women made up 30% of the programming workforce in 1995 (Wootton, 1997). Several participants in this study were asked to explain the causes of this disparity to gain insight into the reasons for the large gender disparity in open source, although few female participants in open source could be located for interviews (See Henson, 2002 for a review of applicable literature).

Open source developers also reported devoting several hours weekly to projects. On average, developers sampled spent a mean of 7.8 hours per week to their main project (with a median of 4) and overall an average of 14.4 hours per week on all projects (with a median of 10). About a third of the respondents are paid for their efforts and they probably explain the 8% of respondents who spend 40 or more hours a week working on open source projects. The "Hacker Survey" identified four primary motivations for contributing to open source groups based on several questions asked of participants, which classified programmers as Believers (33%), Professionals (21%), Fun Seekers (25%), and Skill Enhancers (21%) based on their responses to several questions. Overall, respondents were most often motivated by intellectually stimulating projects (43.2%) which increased their skill base (43.2%). A smaller proportion were motivated by a desire to work with software development teams (20.1%) and to increase professional status (17.4%), providing some evidence for one of the research questions to be explored in this study: that open source groups form a type of "occupational community" for software developers. Only a small proportion of the sample are motivated by "beating proprietary

software" (11.3%) and only 3.4% of the sample believe that open source is "subversive," providing some evidence that most open source programmers are not primarily motivated by an attempt to protest current social institutions or corporate practices.

However, 30% of participants in the 'Hacker Survey' believed in the importance of producing software where the source code remains "open" for the perusal or use of anyone. This wing of the open source movement is more likely to see open source programming as an ideology advancing intellectual freedom rather than a hobby or vocation. Over the past few years there have been efforts to politically mobilize hackers through several organizations including the Electronic Frontier Foundation[9].

D. Chapter Conclusions:

Open source groups have been successful in developing high quality software within a reasonable timeline while generally relying on volunteer labor. Drawing from "the hacker ethic" which has formed the glue for the occupational subculture of computer programmers, open source groups have found tremendous success over the past decade in gaining the participation of highly skilled software developers, although attempts to form business models around open source are in their infancy. The systems of organization that evolve within open source groups as well as the roles that are negotiated between members in the absence of a formal hierarchy form a main focus for this study.

[9] The EFF bills itself as "the first to identify threats to our basic rights online and to advocate on behalf of free expression in the digital age." Their website is www.eff.org

Chapter II: Theory and Literature

A. Chapter Overview:

This chapter provides an overview of past research on post-bureaucratic organizations, and demonstrates through the contingency theory of organizational design, and the more economically-oriented transaction cost approach, how open source groups may actually provide an effective "fit" for the task of software design by solving many of the problems that have perennially plagued the management of software projects.

After an overview of literature on role and task negotiation within post-bureaucratic organizations, the second section of this chapter will consider which shared norms allow members of open source groups coordinate work in the absence of formal roles and motivate members to volunteer their skills. The third section considers long-running conflict between many computer programmers and bureaucratic organizations and systems of control. Finally, this chapter will consider the "open source movement" as a possible emerging "occupational community" which allows participants to improve their programming skills by learning from skilled developers and to gain project coordination and management skills.

B. New Work Organizations in Software Development

1. Role and Task Negotiation in Non-Hierarchical Organizations

According to many observers, modern organizations have shifted from more bureaucratic models of production pioneered by Frederic Taylor in the 1900's to more "post-Fordist" or "post-bureaucratic" structures. While traditional bureaucratic models of labor emphasize top-down control systems, specialization of tasks, and well-defined work roles, "new work organizations" emphasize flatter hierarchies and a minimal division of labor (Castells, 1997; Barley, 1996a; Heckscher, 1994; Touraine, 1974; Bell, 1973). With the growth of technical and "intellectual" labor and other market pressures including globalization, corporations and organizations within a wide variety of industries are becoming increasingly decentralized. For example, some companies have adopted "virtual" organizational forms which decentralize workers and outsource tasks to temporary project teams (Bleeker, 1994). These trends have been especially prominent within high-tech organizations (Barley, 1996a). While companies outside of the technology sector have been slower in adopting "virtual" organizational forms, it has been estimated that 80 percent of Fortune 500 corporations currently are using some form of team-based management for their employees (Joinson, 1999).

Transaction cost economists explain that this shift to post-bureaucratic organizational forms is caused by organizations adopting new governance structures in response to the level of uncertainty within an industry. As the cost of transacting and coordinating with interdependent actors and firms rises, organizations modify the structure of their relations with other firms to minimize opportunistic behavior. According to William Ouchi's

interpretation of transaction cost theory, a governance structure which allows an even greater level of control, "the clan," is required in highly unstable markets. Clans are groups which are integrated by shared norms and are better at coordinating tasks than groups which rely on the market or the top-down decisionmaking of a bureaucracy because they are better able to adapt to changes in market demand and other pressures. Ouchi explains that "clans succeed when teamwork and change render individual performance almost totally ambiguous," because shared norms allow groups to adapt to new problems quickly and to evaluate team performance in a way traditional management cannot (Ouchi, 1981: 84).

Management theory over the last twenty years has come to emphasize the role of trust in creating effective organizations. For example, in "Market, Hierarchy and Trust," Paul Adler argues that as knowledge becomes a more important commodity in markets, firms will adopt high-trust or "community-based" organizational forms. Trust can reduce transaction costs because shared norms and reputations within the community substitute for more formal governance systems, as these normative systems are balanced by the forces or market competition (Adler, 2001).

Shared norms allow workers to manage their own work with little formal oversight, allowing them to adapt quickly to changing product demands, because members of clans internalize group values in the form of principles which they are able to apply to a wide variety of situations and problems. Self-management is especially efficient in situations where tasks are highly uncertain and hard to measure, making them more difficult to coordinate within typical bureaucratic structures. Career paths and work roles in clans are typically non-standardized and lateral relationships, rather than a bureaucratic structure,

link workers and functional divisions. Clans are able to function as self-managing teams because workers are socialized into the same occupational value system. Ouchi explains that "performance evaluation takes place instead through the kind of subtle reading of signals that is possible among intimate coworkers" (Ouchi, 1980: 137). Since members of "clans" share similar value systems, it is less likely that workers will act opportunistically, allowing them to function with little formal managerial oversight, often because clan members police the behavior and attitudes of other members.

The rise of new work organizations in software development firms appears to be an adaptation to the uncertainty of the market combined with the difficulty of managing the software development task. Software programs are notoriously complex and difficult to manage (Curtis, et. al., 1988, Brooks, 1995, Nidumolu, 1995) and the process of design and implementation of a new program is rarely predictable. Most software projects fail to meet their objectives and take longer to develop than expected, and oversight problems are often compounded by a constant shift in customer demand, and by the need to adapt software to work with new hardware. Software projects are also difficult to manage because computer programming is a complex intellectual activity which requires a great deal of technical knowledge and managers rarely have the level of expertise required to effectively coordinate and evaluate programming efforts.

Rating the skill level and efficiency of programmers has always been a problem for managers, both because they cannot determine whether code is high quality and because there is a lack of standard metrics to judge a programmer's skill level. Competing standards include the number of lines of source code programmers have written and the number of bugs in the program, however both standards are problematic because program

size and error rate fluctuate across different programming languages and program type (Albrect and Gaffney, 1983).

Large-scale projects, such as operating systems, are especially difficult to manage because they require a high level of non-routine activities and interdependence between modules of the program (Kraut and Streeter, 1995). In the oft-cited article "Software's Chronic Crisis," Wyatt Gibbs explains that software projects generally fail to meet some or all of their objectives and "some three quarters of all large systems are 'operating failures' that either do not function as intended or are not used at all." In large part this failure occurs because lessons from past projects are rarely carried over to new projects. Unless there is communication between software firms, each firm's access to programming expertise is limited to the experience of their current staff[10]. Gibbs explains that because knowledge is not retained across projects "the vast majority of computer code is still hand-crafted from raw programming languages by artisans using techniques they neither measure nor are able to repeat consistently" (Gibbs, 1994: 87).

Traditional bureaucratic models appear to be too slow to adapt their organization to meet the needs of an uncertain software market. The rise of Microsoft over IBM in the 1980's, for example, was seen by many observers to be the rise of a relatively non-hierarchical or "skunkworks" organization over big business practices. According to many observers, including Microsoft management, Microsoft's success was possible through the adoption of a model where the best programmers become managers, and small project-based teams are used whenever possible (Cusumano, 1998). To survive within the software marketplace, firms must adapt quickly to new product demands in order to remain profitable, placing pressure on software organizations to reorganize

constantly. As a result, high technology firms are plagued by high turnover rates for technicians and "constant restructuring" (Schellenberg, 1996) as firms scramble to remain profitable. The move towards increased employee involvement, neo-craft structures, and network organizations in the software industry appear to be adaptations to a volatile, high-competition marketplace (Lawler, et. al., 1995, Osterman, 1994, Castells, 1997).

Problems of oversight and efficiency in software development are compounded by the inherent complexity of the programming task. Coordination problems are endemic in software development because programmers are generally working on interdependent systems which require a high level of communication between programmers, and there is evidence that formal models of coordination and project reporting generally do not improve team effectiveness in software development (Faraj and Sproull, 2000). Experimental research suggests that coordination problems are simplified when collaborative teams of software developers who share a commitment to the same objectives and high levels of cross-functional communication are used (Jassawalla and Sashittal, 1998, Vann, 2001).

Software development is difficult to standardize and manage because programs are often made up of many interdependent parts, for example a graphical user interface (which displays the inner workings of the program in a graphical format), memory allocation systems, and programs to network with other computers. Since changes to any one part of a program can affect the rest of the program in unexpected ways, programming teams cannot become overly specialized. Rather developers perform best when there is a high level of lateral and informal communication (Curtis, 1988).

[10] See Powell, 1998 on the benefits of interfirm collaboration in intellectual production

Managers who react to these problems by increasing the number of programmers working on a project, rather than improving the managerial structure, may actually make the situation worse, because increased communication and coordination is required within larger software teams. This conclusion is supported by a classic of software management literature, The Mythical Man Month, where Frederic Brooks argues that projects should be led by a single software developer, the "chief programmer," who understands the project as a whole, rather than a traditional manager who delegates all programming responsibilities. Simply adding more developers or "man months" to the project only worsens coordination and communication problems (Brooks, 1995).

According to some researchers, open source groups provide a viable solution to many of the problems of the high-tech marketplace because they use distributed work teams of highly committed workers who self-manage their contribution to projects. This conclusion is supported in "Coase's Penguin" where Yochai Benkler argues that knowledge-intensive industries will be more productive if they adopt "peer production" models such as open source networks, rather than bureaucracies or market structures because they will gain access to a wider pool of specialized labor, as long as mechanisms to ensure work quality and coordination, such as peer review of contributions to projects, are put in place. Benkler explains that "as human intellectual effort increase in importance as an input into a given production process, an organization model that does not require a standard contractual specification of the effort required to participate in a collective effort and allows individuals to self-identify for tasks will be better at clearing information about who should be doing what than a system that does require such specification" (Benkler, 2001: 30).

Contingency theory also provides support for the efficacy of open source groups in designing software. According to this approach, tasks that can be easily broken down and routinized are considered "mechanical," and traditional bureaucratic command and control systems are the ideal fit for managing them. In contrast, more complex or "organic" tasks are best managed by delegating responsibility for decisionmaking to workers and by encouraging a high level of interaction between workers. Organic tasks require a high level of worker self management, especially when there are rapid shifts in customer demand, because the traditional command-and-control managerial system is inefficient in overseeing "complex" labor. As Wolf Heydebrand notes, uncertainty in market demand can lead to organizational adaptations including "informalism, weak classifications [of work roles], loose coupling, interdependence and networking and the propagation of a corporate culture to counteract the centrifugal and deconstructive tendencies of structural flexibility" (Heyderbrand, 1989: 323).

Open source groups which, along with project teams and organizations that adopt matrix structures, share many of the characteristics of the "clan" form (Larsson, 1998) and computer programming appears to be a good "fit" for the complex task of software design. The flat hierarchies and lack of formalized roles and polices evident in open source groups appear to be an emerging post-Fordist model for organizing "intellectual labor." Open source groups appear to be an extreme type of post-bureaucratic organization and they typify the "post-bureaucratic type" as defined (Heckscher, 1994: 334) in the following excerpt:

> Its purpose is mission-oriented and flexible rather than jurisdictional and fixed; its authority structure is based on project teams and task forces, open communication, diffusion of authority, and substantive rationality rather than on a hierarchy involving formal-legal relationships; its

> decision-making style is participatory and problem-centered, with broad delegation; and its careers are characterized by multiple and temporary affiliations, with reliance on subcontracting and experts who have an autonomous professional base.

While the adoption of post-bureaucratic organizational structures may increase worker autonomy, they come at a cost: the certainty of one's role within the organization and the tasks and problems that fall under one's jurisdiction. Workers are often required to take over supervisory tasks such as coordinating work, setting standards for "good work" and ensuring that tasks get completed. The certainty of the bureaucratic hierarchy and policies disappears as workers are increasingly required to negotiate their role and career within new work organizations and to create and enforce group norms (Barker, 1993). Similarly, workers are no longer able to rate their status among other members in the organization by comparing their standing on the organizational ladder because members of new work organizations must negotiate their functional position and organizational status to a much greater degree than in more traditional organizations. Past research has shown that role and task negotiation has been especially important when new technologies, such as gastrointestinal endoscopy and radiological imaging enter established occupations (Barley, 1990, Zetka, 2001) or when new occupations such as computer programming emerge (Kraft, 1977).

The move from bureaucratic organizations to flatter "new work organizations" requires new systems for control and commitment of employees. While bureaucratic systems control tasks by creating functional divisions of labor which focus on well-defined tasks, organizations that lack a command structure must rely on normative or "concertive" control to motivate and monitor employees. Often "self-managing teams"

are used where employees are expected to govern the behavior of their teammates, and workers must negotiate roles and status within the group in the absence of fixed hierarchies and specialty areas. This can be accomplished through the use of normative systems which reward adherence to group priorities (Morrill, 1991) and by the evolution of reputation systems which use informal peer evaluations to rank worker skill level (Heckscher, 1994:27).

Despite a cornucopia of suggested project management models for software development, the computing industry still lacks a dominant model for managing programmers, which may be due in part to the difficulties of "placing" programmers and other technicians within the organizational hierarchy. Barley (1996b) finds that non-technical workers are often uncertain about whether they should regard technicians as skilled servants or as equals. He outlines two emerging roles for technicians: buffers who provide information in an understandable form, allowing another occupation to do their job, and brokers who bridge different occupations by translating the information that passes between them. These role uncertainties combined with managerial encroachment can lead to job dissatisfaction among technical workers, and high employee turnover (Zabusky and Barley, 1996). Organizations have reacted to the problem of "placing" programmers through a variety of methods. At the dawn of computing, programmers were generally treated as clerical laborers, mechanically implementing plans devised by managers and systems analysts. While modern programmers have gained some degree of occupational autonomy and are generally highly paid, there are still struggles against the routinization of the programming task[11].

[11] See Chapter 2, Section C1.

It appears that, at least in technical labor and other emerging occupations, workers must actively negotiate their identity within organizations to gain occupational autonomy. This negotiation of professional identity was examined in Stephen Barley's study of the effect of new imaging technologies on the work roles of radiologists. Barley (1990) found that traditional radiologists and new-technology radiologists negotiated new social roles in response to the introduction of new ultrasonic imaging technology. Barley mapped interaction networks of each worker in an attempt to determine the types of subjects that they discussed with others to determine "collegial relations" among the radiologists. If radiologists often discuss a "variety of work-related issues," then they have "strong" bonds. Mapping these bonds, and how they changed with the introduction of new technology, provided one measure of renegotiated roles within the occupational groups. Finally, an ethnographic analysis of the actual content of these interactions provided some insight into whether a new "interaction order" was formed.

Barley found that the established social order within the organizations under study was threatened by the introduction of the new radiologists, and there was a need to negotiate a new order. Barley found that new-technology radiologists were not considered a threat to the "old guard" who typically had strong diagnostic experience that the new radiologists lacked, allowing both groups to educate each other to their mutual benefit. The mid-tenure radiologists were the most threatened by the changes because they were not strongly competent in either new technologies or diagnosis, and found difficulty negotiating a new occupational identity, while the high and low-tenure groups were able to negotiate new roles that were mutually non-threatening.

Despite their expertise in interpreting new-technology images, the technologists were careful to "establish a social order" which did not threaten the expertise of the radiologists. It appears that computer programmers and other IT programmers must also negotiate their identity within organizations where the level of prestige and organizational role of programmers occupation is not well defined (Pettigrew, 1973a).

For many open source participants, association with their peers in a relatively non-hierarchical environment may provide some release form the pressures of adapting to roles in traditional organizations. In open source groups, status is typically determined by an individual's reputation among his or her peers[12]. These groups are not completely free of the trappings of status, however and, especially within larger groups, there may be several levels of status. While anyone can monitor the mailing lists (which provide news about the project, specific changes and problems, etc.) or submit patches for possible inclusion in the kernel, only "committers" can make changes to the source code itself[13]. In addition, larger groups typically have a small group of committers or a leader who helps to coordinate major changes and to resolve disputes. Status is earned when members demonstrate technical expertise thorough their comments on mailing lists and when they make useful contributions to the project[14]. Open source developers compete to become committers within high profile projects, and successful contributions to one of these projects can help their reputation and career, as projects typically list all of the committers and leaders on a "contributors page." Software firms often recruit open

[12] For example, the technically-oriented web community of Slashdot uses a fairly complex rating system where user's comments about an article are moderated and the sum of these moderations, positive or negative is that user's "karma." Similar rating systems are used in on-line open source communities such as SourceForge and FreshMeat, while other groups simply rely on peer reputation to rate programmers.
[13] The process of becoming a committer and the extent to which the group functions as a meritocracy will be examined in the two case studies of Star Linux and LithiumBSD.
[14] Based on evidence from preliminary interviews.

source programmers with high status in the community. According to some observers the open source community functions as a rating system for programmers, as the best programmers gain recognition from the community (Stalder and Hirsch, 2001).

This reputational system may help to solve a problem endemic to complex and technical labor: determining a practitioner's level of expertise with different tasks. Peer rating systems have been used with some success in the scientific community, for example, where a scientist's expertise is evident by the respect of her peers even if the research itself is too complex for most observers to judge. One possible solution to dilemmas in software development, including rating skill levels and coordinating complex tasks, is for the occupation to adopt a craft model of production which eschews formal divisions of labor and hierarchies in favor of a more community-based labor model where expertise determines formal status among one's peers[15].

In this study, I observe the process of role and task negotiation within open source groups. Since groups lack a formal hierarchy beyond a simple in-group/out-group criteria and elected leadership roles, it is expected that group members will have to define their own role in the group and be self-directed in determining which projects or tasks to pursue. While traditional "command and control" organizations can motivate their employees with the "carrot" of salaries and raises or the "stick" of demotion or firing, open source groups must rely on systems of control (e.g. normative control) which motivate technically-skilled volunteers to contribute their work to a project. Ethnographic study of the organization of work in a non-hierarchical organization has been a useful method for uncovering the systems of coordination and control which organize technical and other workers within post-bureaucratic organizations and add to the literature on

management of complex, intellectual work through the encouragement of worker self-management and coordination of tasks.

2. Control and Commitment: The Ideology of Open Source

As responsibility is shifted from managers to workers and as organizational ideals are internalized as "concertive control," workers learn to police their own work and the work of their team. This may be the most effective system of control for "organic" tasks, such as computer programming which cannot be easily "deskilled" or supervised under traditional management systems (Braverman, 1998). Concertive control can either be encouraged by management or can emerge from the workers themselves. However, overt managerial efforts to encourage normative commitment risk creating cynical employees.

In Engineering Culture, Gideon Kunda describes an effort to "engineer culture" in a high-technology firm where team mottos (e.g. "quality is job one"), company rituals and cultural advocates were employed to socialize workers into company norms with mixed results (Kunda, 1992). It appears that the most effective and all-pervading systems of concertive control are generated by the workers themselves (Barker, 1993).

Morrill (1991) describes the negotiation of a new normative system of control between workers when he examines the negotiation of roles and conflicts by executives in a toy company that adopted a matrix organization. Morrill found that after the matrix system was adopted, executives lacked the norms of conflict resolution which were imposed under the earlier, more traditional system, where senior managers would quickly resolve disagreements in informal meetings. Under the new matrix system, no single

[15] See Section C2.

manager had the authority to enforce such a decision. As a result, a system of roles and rituals evolved at Playco, which allowed managers to settle disagreements through colorful "shoot-outs" between executives who followed well-defined "rules of engagement." Individuals who refused to "play by the rules" negotiated by the group, either through avoidance of conflict or aggressive tactics, were censured by the group. Morrill concludes that organizations, which lack a formal system for determining roles and settling conflicts, will negotiate and enforce a new symbolic order.

Open source groups also lack a formal hierarchy and appear to organize tasks and work roles through normative control in the absence of formal systems of coordination and roles. The primary status division in these groups is between committers (those with access to make changes to the source tree), and non-members who only have the option to submit patches to committers who choose whether or not to incorporate them into the source code. And while there is a relatively flat hierarchy between committers, members with higher status are better able to influence group decisions, including which priorities should be the focus of the group[16]. Just as workers in Playco developed a normative order to resolve conflicts and define worker roles, it appears that open source groups reward adherence to a governing normative order[17]. A preliminary indication of these norms is evident in the literature of open source advocates.

While there are a variety of reasons for programmers to participate in open source, (Boston Consulting Group, 2002) one of the leading proponents of open source development, Eric Raymond, provides a popular exposition of the norms that hold together the community in his essay The Cathedral and the Bazaar. Raymond explains

[16] Based on evidence from preliminary interviews.
[17] The specific norms that guide the open source community will be examined in the two case studies.

that open source groups develop when programmers want to solve problems that interest them and when other developers share that interest. As a result, a community or "bazaar" of specialists develops around the project which is more efficient than the top-down "cathedral" approach of management (Raymond, 2002).

Other observers have identified a new work ethic evolving in open source and other technical jobs which is displacing the Protestant ethic: the "hacker ethic." Pekka Himanen identifies the values of this new work ethic which include "passion" which is an "intrinsically interesting pursuit that energizes the hacker and contains joy in its realization" and "social worth and openness" where hackers "want to create something valuable to the community and be recognized for that by their peers" (Himanen, et. al., 2001:139). Members of the open source community rarely pursue short-term gains in their projects, such as maximizing profit from their project. Rather, they tend to participate when they find a problem intriguing or to further the growth of non-proprietary software. The rise in importance of the "hacker ethic" and the decline in traditional work rewards can help explain the rise in open source participation among IT professionals and others. While this ethic has held together the occupational community of programmers, starting from the development of the first computer (Levy, 1984; Kidder, 1981; Weinberg, 1971), only with the rise of the Internet has the creation of a trans-local community of programmers become possible (Ljungberg, 2000).

Observers of non-hierarchical organizations have found that, in the absence of well-defined identities which define the level of prestige of group members, alternative systems evolve including gaining status according to a members the reputation among occupational peers, and their adherence to group norms (Fine, 1984). It is not initially

clear which forms of normative control hold together the two open source groups under study (Star Linux and LithiumBSD). It is possible that both groups are held together by a similar form of normative control common to most open source groups, or that each group has evolved a unique "normative order." This study adds to the literature on normative control in post-bureaucratic and technical organizations pioneered by Barley (1990), Morril (1991), and Kunda (1993).

C. Administrative Control and Software Development

1. Managerial Attempts to Control Programmers

From the dawn of the computer industry, managers have attempted to make the complex task of programming more manageable by deskilling the task of programming. One solution was to create a "dual ladder" system of labor (Kornhauser, 1962) which, if successful, allow managers a greater degree of control over an occupation or profession by standardizing knowledge and tasks. Early observers of the computing industry described the division of "data processors" into two categories: "systems administrators" who were responsible for the actual design of new programs and had the highest status in the profession, and "coders," with the lowest status, who actually implemented the plan created by the systems administrators (Greenbaum, 1979; Kraft, 1979). New coders were recruited from backgrounds that varied from mathematics PhDs to secretaries already on the payroll, and inhouse training was often preferred because outside experts and university trained programmers were typically more costly and more difficult to manage.

In his classic study 'Programmers and Managers: The Routinization of Computer Programming,' Philip Kraft argues that from the very beginning managers have attempted to implement reforms that limit programmer autonomy in order to retain their traditional supervisory role (Kraft, 1977). Andrew Pettigrew describes this conflict between managers and programmers in his classic study of a traditional firm, Brian Michaels, which implemented technical systems within a non-technical organization. The introduction of a computer system into the firm created conflicts between programmers who were unsure of their organizational status and managers who often felt threatened by their lack of understanding and control over technical systems (Pettigrew, 1973). A variety of strategies were developed by programmers to retain their autonomy within the organization in response to managerial attempts to gain control, including protective myths, norms that denied the competence of outsiders, norms of secrecy, and protection of expertise through control over training and recruitment of new programmers. For example, computer workers at Brian Michaels would routinely overestimate the time required to complete work to and use vague language when explaining technical issues to managers. Computer workers also relied on physical isolation from managers and out-of-the-ordinary work hours to decouple their work from managerial control (Pettigrew, 1973: 146). None of these protective strategies are as effective in the modern marketplace where technical skills are much more common for both managers and workers, but it is clear that the conflict between programmers and managers is endemic within the computing and other industries. The persistence of this conflict is clear both in repeated attempts by managers to standardize the programming task, and the success of "techie

humor," such as the Dilbert comic strip, which belittles the abilities of traditional managers of technical projects.

Other recent and historical attempts to standardize the task of programming include the development of "structured programming," the COBOL programming language, and Microsoft's Visual Basic language, which are attempt to make the process of programming transparent to managers and other non-programmers. These developments attempted to break down the task of programming into easy to understand steps which are accessible to managers, ideally allowing them to program without relying on the expertise of programmers (Kraft, 1979).

While attempts to standardizes programming generally have failed to routinize the programming task, these changes have created new systems of control over the programming task which are incorporated into the programming process itself, simplifying many routine programming tasks and making programs more readable. Programmers engaged in more complex tasks have best resisted these efforts at standardization. According to some observers, the "dual ladder" structure, which evolved during the rise of computing, still exists within the software, dividing programmers into two distinct professions (Burris, 1993, Ensmenger, forthcoming, Abbott, 1988:38).

The endemic conflicts between managers and programmers described by Philip Kraft are still prevalent today (Kraft, 1977). In an in-depth study of the development of Microsoft's NT operating system, Pascal Zachary describes managerial practices that slowed down the complex process of software development. Despite efforts to allow developers a great deal of autonomy over decisionmaking and the hiring of managers with high levels of technical expertise, the project suffered from "feature bloat" as upper-

level managers continued to add features to an already late project, without a real awareness of the difficulty of implementing them (Zachary, 1994).

This conflict between managers and programmers can be explained as an instance of the conflict between the “administrative principle” explicated by Weber in his writings of bureaucracy (Weber, 1922) and the “occupational principal.” The occupational principle emphasizes an individual’s standing within the community of her occupational peers over her position within a bureaucratic hierarchy, and tacit over formal knowledge (Hall, 1968; Freidson, 1973). This tension is evident in the history of factory production of goods, which often started as a craft as tradesmen passed their skills down to apprentices. With the rise of the assembly line and work-practice studies pioneered by Frederic Taylor, it because possible for managers to gain control over this tacit knowledge by routinizing it. With the rise of the assembly line, automobile workers lost their autonomy and were absorbed into a administrative structure that could control assembly and make it more efficient.

A similar conflict between an occupational orientation and administrative control occurred within the engineering profession. Peter Meiksins and Chris Smith examined the structure of the engineering profession in several countries and concluded that engineers rarely attain occupational autonomy because "the professionalization of engineering would mean that a class of employees on whom employers depend for the day-to-day functioning of the firm, to whom they are obliged to delegate a degree of responsibility, would become extremely expensive, scarce, and difficult to control" (Meiksins and Smith, 1993:140).

Historically, engineers, like programmers, were often torn between identification with their occupation and loyalty to their employer, and between a professional or craft orientation towards their work. For many professions, including engineering and health support occupations, there was little resistance to inclusion within a larger administrative structure because it provided benefits including routinized work conditions and promotion systems, and a regularized income. Other professions resist inclusion within bureaucratic systems by pursuing professional status based on the claim that they require autonomous decisionmaking and control over entry into the profession for the public good (Friedson, 1973:19), or conversely by operating under a craft system of production which emphasizes flat hierarchies and a relatively low level of task specialization. It appears that the open source movement may be a shift away from traditional models of professionalization, which include control of entry into the profession, adoption of a system of ethics and formalization of a body of specialized knowledge, in favor of a neo-craft model of production. This move away from traditional models of professionalism appears to becoming something of a global phenomenon as professions as established as doctors and lawyers are increasingly working within organizations. As Richard Hall explains, as professions move away from the solo practitioner model towards models which allow increasing control over the professions by clients and organizations, "we can expect 'looser' organizational structures as clusters of experts are brought together for problem solving and the generation of new ideas" (Hall, 1974: 332).

At the same time, with the rising importance of technical and specialized knowledge in the world economy, technical occupations are gaining increased professional autonomy. Freidson (2001) describes an ideal type where professionalism, rather than

bureaucratic or market models, organizes the economic sphere. In this model, worker control of occupations, and the economy as a whole, is possible as "each specialization controls the work for which it is competent, negotiates its boundaries with other specializations, and by that method determines how the entire division of labor is organized and coordinated" (Freidson, 2001: 55). It is possible that the open source movement is an instance of professional self-definition as programmers organize voluntary associations, which allow them to contribute their skills to peers who are able to appreciate their contributions.

In some ways the open source movement can be seen as a challenge to the legitimacy of traditional management practices in software engineering and as the rise of a craft-based model of programming[18]. By producing software that is comparable to proprietary software, and often more quickly, the open source community demonstrates that computer programmers can effectively self-manage software development. Even if programmers continue to work within organizations rather than become completely independent by starting their own organizations, success at open source projects provides a strong argument for increased occupational autonomy for programmers by showing that they can work successfully without formal oversight.

The open source community also allows programmers to evaluate and improve their programming skills through peer evaluation of their programming efforts. As programmers come to identify with their occupational community rather than the organizations that employ them, they are more likely to gain marketable skills, and computer programmers as a whole are more likely to become an autonomous occupation (Gouldner, 1957). And as participants in open source communities develop high-quality

software and gain respect within the software industry (Thompson, 2000, Hammel, 2001, Markoff, 2002), they challenge the legitimacy of traditional model of project management and promote an alternative development model: "peer development" coordinated by the workers themselves.

Through study of this “emerging occupational community,” I attempt to uncover an alternative model of professional mobilization and professionalism. While managerial control in the workplace is not a primary focus of this study, members of open source groups were asked for their opinion regarding attempts to standardize programming practices and whether they participate in open source groups to gain control over the product of their work which they lack in traditional workplaces.

2. An Alternative Model of Professionalism: The Occupational Community

Despite attempts to professionalize computer programming (Ensmenger, 2001; Meyers, 1997) most technical work does not adhere to the traditional models of professionalism or craft (Barley 1996b). Early functionalist models of professionalism focused on the “niche” that different professions filled within larger society and emphasized that professions which best meet the current needs or niches in society become established.

Professions that fill these niches then develop a formal body of scientific knowledge, an educational system to train new workers, and control over entry into the occupation through licensing, among other characteristics. Later elaborations on this model added an attitudinal dimension to this model, arguing that professions develop unique ideologies

[18] See Chapter II, Section 2 for more on inter-occupational conflict.

that are internalized by their members (Hall, 1968). Occupations can become full professions, according to this model, if they meet these formal prerequisites (developing a system of education, etc.) as well as a "calling" towards their work and a system of ethics which defers immediate economic gains to pursue the occupation as an "end in itself" (Mills and Vollmer, 1966; Bourdieu, 1996).

The conflict approach to professions challenges the functionalist view by chronicling the battles between competing professions, such as the battle between obstetricians and nurse midwifes, for legitimacy as a profession. In this model, rather than fitting a preexisting niche within society, professions must work actively to carve out their identity by mobilizing political and cultural resources or by gaining control over the socialization of specialized knowledge (Hall and Engels, 1974).

A final approach to professionalism related to the conflict model will be emphasized in this study: profession as a process. While earlier models of professionalism attempted to identify characteristics which differentiate professions from "semi-professions" (Hall 1968), there has been increasing recognition that a single ideal model of professional criteria that any occupation can follow to become professionalized ignores the wide diversity that exists between occupations that are considered "professions" (Atknis, 1983; Davies, 1983).

Rather than forming explicit functional niches, according to critics of unitary models of professionalism, professions evolve in response to particular historical and societal changes, and the formation of a new profession can only be understood through study of the historical circumstances where it arose (Abbot, 1988; Perkin, 1996). Similarly, other students of professions have examined the importance of creating and maintaining public

"legitimacy" for a profession by mobilizing cultural resources. Rather than having one unique "niche," professions may adapt their professional image to defend their professional standing in response to social and economic changes, or threats to their legitimacy. A final model adds to the historical model of professions, emphasizing the history of inter-occupational conflict.

Atkinson (1983) explains that rather than having a unified professional ideology, professions often include groups with very diverse interests and professional "missions" that they may attempt to persuade the profession to adopt. In periods of change it is possible for "missionaries" within a profession to "stake out claims for new territory, or to re-colonise abandoned ecological niches" (Atkinson, 1983: 240). Similarly, Bucher and Strauss (1961) explain that professions are divided into "a loose amalgamation of segments" which "tend to take on the character of social movements" as they attempt to further their own interests by reshaping the profession through the workplace and educational institutions (Bucher and Strauss, 1961: 333, 325). It appears that the "open source movement" may be an instance of this type of mobilization of a new professional identity for a faction of computer programmers.

Although computer programmers and other high-tech workers generally employ abstract knowledge and must acquire a large body of expertise, both qualities that would suggest professionalism, these occupations lack control over mean of entry to the occupation, and training is frequently gained on-the-job, rather than primarily through formal education. Students of professionalization have had difficulty determining whether computer programmers are a profession, a "semi-profession" or a craft[19]. Divergent visions of the occupation have existed since almost the beginning of

computing. Each of these occupational models can be seen as a competing paradigm which various interest groups promote, a common situation within emerging fields such as high-tech industries (Freidson, 1973: 31, 93).

On the one hand, some believe that programmers should become professionals through the traditional route: by controlling entry into the profession and building a formalized and abstract body of knowledge imparted to students by accredited schools. An alternative but similar model is offered by certification courses which allow workers to prove their expertise in a limited area of programming, such as database management or graphical interfaces, without the time required to gain a general university education. This model has been historically supported by employers suffering from a shortage of computer programmers and by workers who desire quick entry into the occupation (Ensmenger, 2001).

While both of these models offer a certain level of freedom from administrative control, professionalization also acts as a system of control over occupations, by standardizing work methods and education (Freidson, 1973). According to some observers of these competing occupational models, managers of computer programmers actually encourage some level of professionalism because it made programmers easier to control by avoiding unionization and other "unprofessional" forms of worker resistance (Kraft, 1977).

A third model of professionalization for software developers also exists: programming as a craft. While the other two occupational models emphasize formal knowledge and professionalization of programming, a craft model values tacit knowledge. Workers gain tacit knowledge on the job or from other workers, and craft-

[19] See Ensmenger, 2001 for a historical view of these trends.

based occupations restrict entry to individuals who evince practical expertise and normative commitment to the occupation, while devaluing formal credentials (Van Maanen and Barley, 1984). Craft-based models of labor can function as an alternative method of judging worker expertise, which is communicated by a worker's reputation within the occupational community. This model may be an effective method for organizing the occupations, as reputation-based model of occupation are especially successful when the quality of work is difficult to evaluate (Stinchcombe, 1953).

In his study of alternative labor models, Arthur Stinchcombe argues that that craft or "occupational" models of labor can provide "a functional equivalent of bureaucracy" because the "characteristics of the work process are governed by the worker in accordance with the empirical lore that makes up craft principles." Stinchcombe explains that these normative controls "are the content of workers' socialization and apply to the jobs for which they have preferential hiring rights" (Stinchcombe, 1953:170). In occupations reliant on worker expertise and skills that are difficult for the non-expert to evaluate, peer evaluation of work quality may provide a better system of oversight than managerial control, without the costs of a formal administrative structure. Eschewing bureaucratic regulations which define employment practices, craft-based occupations typically follow occupational principals, taught through socialization into the occupation, which are applied by the worker to specific problems (Heckscher, 1994). In the craft-based occupational model, clients and organizations are expected to give workers a high degree of discretion in determining the best way to solve work-related problems.

This third model of professionalization can be seen as a type of "occupational community." These communities are groups of individuals with similar skills and

expertise unified by work cultures which "consist of, among other things, task rituals, standards for proper and improper behavior, work codes surrounding relatively routine practices and value of these rituals, standards and codes" (Van Maanen and Barley, 1984:324). Unlike a membership in a traditional bureaucratic organization, members of occupational communities identify strongly with their occupation, and a "culture of achievement," which lends status to those with more expertise rather than the "culture of advancement" common to most organizations (Trice, 1993). Distinct patterns of values and beliefs surround craft-based occupations, their work activities often carry over to leisure time, and workers tend to identify with others within their occupation rather than with the organization (if any) that they work within (Van Maanen and Barley, 1984: 302, Pettigrew, 1971). For example, technically proficient "hackers"[20] within companies often eschew traditional schedules, business dress codes, and other organizational norms.

Hacker values include a distrust of opinions that they cannot validate themselves and a desire to pursue interesting problems, even if there is no immediate reward for doing so (Himanen, 2001). There is also evidence of intra-occupational division within the computing industry itself. Many occupations in computing are aligned with managerial norms and interests, while programming tends to be more occupationally oriented[21].

Membership in an occupational "community of practice" provides a forum where members acquire tacit knowledge and are socialize into "ways of doing things, stories, gestures, symbols, genres, actions, or concepts that the community has produced or adopted in the course of its existence" (Wenger, 1998: 83). In these communities, craftsmen appreciate the handiwork of their peers, even if people outside the occupation

[20] See Glossary for definition
[21] See Hebden, 1975 for an earlier study of intra-occupational divisions within computing

consider their work mundane. Appreciation of good craftsmanship is evident in programmers who examine code to find different ways to solve problems because programmers often have different "coding styles."

This appreciation of the craftsmanship of skilled programmers is evident in a speech by an early recipient of the Turing Award, which is given to individuals noted for their contribution to the field of computing science. Donald Knuth stated that "my feeling is that when we prepare a program, it can be like composing poetry or music . . . programming can give us both intellectual and emotional satisfaction, because it is a real achievement to master complexity and to establish a system of consistent rules. Furthermore when we read other people's programs, we can recognize some of them as genuine works of art" (Knuth, 1987:39).

It appears that the open source community functions in a similar way by providing peer review of programmer's efforts and reputational measures of the best coders in a way not possible within a traditional model of professionalization, due to the difficulties of quantifying good programming (Stalder and Hirsch, 2001). Open source groups appear to be one way for programmers to participate in a community of practice[22], and for many programmers the open source community functions as a place to refine their skills (Boston Consulting Group, 2002).

Programmers not connected to the occupational community risk being left behind by market shifts which constantly demand new programming skills to gain the best jobs. As a result, a craft-based model of programming may provide a better occupational model

[22] The growth of the Internet and other forms of electronic communication have fueled the growth of communities of interest, including occupational communities. Most open source groups use a wide range of communication channels including email lists which focus on different aspects of the project, IRC channels, instant messenger software, which allow distributed groups to communicate easily. Email,

for keeping up with new skill demands, and may act as a training ground for the more complex and difficult-to-standardize tasks involved in programming, which are not taught as well in formal systems of education[23]. Rather than learning occupational skills and norms through formal education, craft-based models of socialization emphasize on the job training and experience with practical problems.

It appears that, for many programmers, participation in open source projects with other skilled programmers may provide an alternative model of socialization into the programming occupation and refinement of skills. In contrast, most occupations rely on the traditional professional model of socialization through formal education and other institutions that provide members of an occupation with similar experiences and worldviews. Similarly, open source groups appear to fulfill many of the needs of traditional professions including a sense of calling to the occupation, and the ability to make decisions autonomous of pressures from clients or employing organizations (Hall, 1969; Freidson, 2001). In this study I attempt to determine whether open source groups act as a new form of professional association and "occupational community," possibly providing a model for other technical occupations. Through observation of group activities and interviews with members, I attempt to determine the level of importance open source groups have in shaping the careers and professional skills of their members and to what extent group members have adopted a traditional model of the programming profession.

D. Chapter Conclusions

---

mailing lists and other forms of electronic communication make it possible for globally dispersed workers to function as project teams and "virtual organizations" (DeSanctis and Monge, 1999).

The sections above provide an overview of the literature on normative control and role and task negation in post-bureaucratic organizations and a summary of the problems plaguing the majority of software development efforts. The "occupational community" that appears to have evolved within open source groups is suggested as one possible model for allowing worker self-regulation of complex tasks which are otherwise difficult for typical management structures to effectively oversee.

The systems of normative control that evolve in these groups will be examined as "negotiated orders" created as group members with varying skill levels attempt to gain status and carve out preferred roles within the group. It appears that learning role negotiation is especially important both for technicians who often lack a clear role within organizations (Barley, 1996b) and for any worker in a "project-based" or post-bureaucratic organization. Open source groups may then function as a training ground for technical workers to thrive in an evolving labor market where credentials and job titles are less important than a worker's reputation among her peers and her ability to negotiate work tasks within a project team.

While there has been a great deal of attention in organizational studies on the rise of post-bureaucratic organizations, there is a relative dearth of ethnographic work examining how work roles and task allocation are negotiated within systems that cannot make decisions by fiat. Open source groups typify many of the characteristics of the post-bureaucratic organizational model: there is little or no hierarchical structure[24] and

[23] See Analytical Note for greater detail on open source participation by computer science students.
[24] The leader of a project may retain final say over any changes to a program, but members always have the option to take the source code and start their own group. This tends to encourage a consensus based model of decisionmaking

members typically work autonomously, managing their own contribution to the project. This organizational model both resolves difficulties inherent in managing the programming task and can lead to problems including frission over who controls decisionmaking in the group and who should perform a given task which would be minimized under a more bureaucratic system. The process of role negotiation and task coordination will be observed in two case studies of open source groups to gain further insight into the functioning of post-bureaucratic organizations

Chapter III: Methodology

A. Chapter Overview

Qualitative data was collected from two open source groups, 'Star Linux,' a 'distribution' of the Linux operating system and 'LithiumBSD,' a distribution of the BSD operating system. To gain a clearer picture of the organizations as a whole, I sampled several channels of information within each group. Channels of communication that were sampled include email lists and archives, interviews with current and past members, documents and manuals, and IRC channels, to gain a well-rounded understanding of organizational structure, culture and roles[25], and to increase the validity of research findings (Pandit, 1996). To some degree, research questions and methods were developed in the process of collecting and analyzing data, in concurrence with grounded theory methodology, which was used to determine sampling and to develop tentative hypotheses. Tentative research questions were also suggested based on information gained from fifteen interviews of open source developer and users that formed the "preliminary sample."

[25] See glossary for definitions of the channels of information to be sampled

B. Research Agenda

1. Research Questions and Brief Summary of Results

a. Role and Task Negotiation and Commitment

The evolving norms of the open source community were explored to determine how individuals gain status and recognition within the group, as well as the system of normative control that coordinates contributions to each project in the absence of a formal hierarchical structure. Since open source participants belong to groups that do not award status based on past education or employment[26], I expected to find that a member's apparent contribution to the project would primarily determine status within the group. "Quality of contribution" is not a simple objective standard, because members of open source groups have competing standards for what "good" programming is and which priorities the group should adopt. This suggests that, to some degree, persuasion of other members and subjective criteria help determine whether a proposed solution is implemented within the group.

I expected that members with the highest status were individuals who are best able to persuade other members that their solution is the "best" when challenged by competing solutions, as long as their ideas are not obviously incorrect. The negotiation of technical and personal conflicts between members were examined to determine how a "negotiated order" is established and maintained within the group, and whether adherence to these shared values determines status to a greater degree than a member's technical skills. Based on data collected in this study, I found that members must make an effort to be

visible within the organization to gain "commit" and leadership status, and to institute changes within the organization. Generally, programmers who do not make an effort to be noticed within the group do not gain high status, especially when group size extends beyond a few dozen members.

According to one interviewee in the preliminary sample, visibility is not earned simply by writing the best code, in part because it is easy for an individual's patches to be ignored due to the sheer volume of contributions to the group. Rather, to become respected within the group, developers must make an effort to communicate their solution to other members through mailing lists, often repeatedly, and be willing to defend their ideas when criticized. Rather than allowing their programming efforts to speak for themselves, high profile members are often both active self-promoters and known for their witty rebukes in the face of criticism. The process of gaining visibility within the group was examined by interviewing members with different levels of status within each group and by observing the groups in action.

b. Managerial Control and Occupational Autonomy

Employed interviewees were questioned about their satisfaction with their current workplace, and how that structure compares with the open source group they participate in. I found that many open source developers participate because their current employment does not give them a high degree of autonomy or the opportunity to work with a community of their peers on projects that interest them[27]. All interviewees were

[26] Based on evidence from preliminary interviews.
[27] See also, Boston Consulting Group, 2002.

also asked about their ideal work situation and if not currently employed, where they plan to work next to get an idea of their future career plans and whether their open source experiences are helping them to develop their future career. Many members answered in the affirmative, citing their current employment by open source corporations, and more traditional organizations that use open source tools.

Interviewees were also asked about their level of support for a traditional model of professionalism, including questions about whether credentials, degrees, or practical experience should be preferred in hiring of new programmers to determine if they value tacit over formal knowledge, as would be expected within an occupational community. It was found that committed members of open source groups were generally dissatisfied with traditional programming practices used in most workplaces and taught by colleges, as well as bureaucratic managerial structures which typically oversee software development, although a few had been lucky to work under 'clueful' managers who allowed programmers a high level of control over their work practices.

Participants were asked about the norms that govern their community and the consequences of not conforming to them. Generally, it was found that in open source groups, work is organized by general principles which are interpreted by workers when they face specific work situation rather than formal rules. It appears that members of open source projects are rewarded with high status both to the extent they conform to the normative rules of the group and that their programming efforts are deemed a valuable contribution to the project.

C. Methods and Data

1. Methods

The primary methods used in this study were in-depth interviews of current and past group members, and participant observation of group activities through various channels of communication. Since the organizational activities of these groups is conducted online through email lists and IRC channels, it was possible to "participate virtually" in the organization by monitoring relevant channels of electronic communication (Markham, 1999).

The primary advantage of this dual-method approach was that participant observation and interviews can complement each other by filling in informational gaps and biases that exist in each method when used separately. For example, statements made in interviews about an interviewee's actions and attitudes were verified by examining their later actions and conversations. Similarly, interviewees provided detail and contextual information about events and statements recorded during participant observation (Becker and Geer, 1970).

This data was examined to understand the process of negotiating work processes and roles, including coordination of work and task assignment. Content analysis was also be used to examine organizational "artifacts" including emails, IRC sessions, and documents, to identify important themes and issues in the group as well as group values, and insights from these interviews were used to formulate interview questions. Initially, a large amount of material within each of these communication channels in the group were sampled and gradually the focus shifted to specific incidents, subjects of discussion,

group members and subgroups, in part as a method to make the processing the vast amounts of intra-group communications more manageable. Theoretical sampling principles were used to determine which individuals and issues within the group were of the most interests for detailed study.

2. Sampling Design and Population

a. Preliminary Sample:

The preliminary sample consisted of fifteen interviews of members of open source groups and users of open source software which were analyzed to gain a general understanding of how open source groups function, and to discover possible research questions to study. The population included users and developers of open source software, such as the Linux or BSD operating systems. The characteristics of interviewees actually selected are listed below. A summary of findings from these interviews can be found in section 5-D.

Table 1: CHARACTERISTICS OF PRELIMINARY INTERVIEWEES

Interview Number	Employment/Student Status	Employer Type
1	Full-Time Systems Administrator	Small Organization
2	Full-Time Systems Administrator	Large Non-Profit
3	Student, Not Employed	
4	Student, Summer Employment for University Department	
5	Student, Summer Employment	Several small firms
6	Student, Internship/Part-Time	Government Agency
7	Student, Not Employed	
8	Student, Not Employed	
9	Full-Time Systems Administrator	Government Agency
10	Full-Time Systems Administrator	Government Agency
11	Full-Time, Senior Programmer	University
12	Full-Time, Computer Lab Director	University
13	Full-Time, Software Developer	Large Software Firm
14	Full-Time, "Corporate Advocacy"	Large Software Firm
15	Full-Time, Programmer	Mid-Sized Open Source Firm

b. Primary Sample:

Members of two large open source groups working on operating systems were sampled: Star Linux and LithiumBSD[28]. From each group, twenty members with varying status and areas of technical specialty were interviewed[29] and theoretical sampling methods were used to determine which categories (e.g. types of roles or levels of status) to further sample (Glaser and Strauss, 1999). Two local members of the LithiumBSD group from the preliminary sample were interviewed for further data and acted as informants regarding group practices. I attempted to gain legitimacy within these groups

[28] Both names are pseudonyms
[29] Both groups provide a full list of committers by status and specialty area.

by having these informants vouch for my identity and credibility, and on occasion, interviewed members of the group invited other members to participate in my study. A local member of the Star Linux group who was interviewed in the preliminary sample also provided some basic help in identifying further interviewees.

Several channels of group communication were sampled including email lists, which act as the primary means of communication for most members (See Table 2a). There are generally dozens of lists within a large open source group, and each functions as a specialized forum for a specific topic, such as bug reporting, or type of group member, such as leader, developer, etc. The main lists were monitored are lists intended for developers and members of leadership groups and all general discussion lists. Email archives provided a valuable history of past communications and they were sampled to follow important issues over time. These email archives are also searchable, and key terms (such as 'disputes'), names and events were searched within applicable mailing lists. All general email lists in both groups were tracked. A subproject within each group were studied in greater detail (See Chapter VIII), and archives for these groups going back several years were read as well.

The second source of information that was sampled was the IRC channels for each group. These channels are "chat rooms" which are focused on specific topics, much like email lists, but which allow for immediate conversations between anyone who is "logged in" to the room. Large open source groups generally have one or two primary channels, which may contain hundreds of individuals communicating simultaneously as well as specialized forums intended for technical help and other subjects, within each of the

several IRC networks[30]. Many hours of on-line communication within these channels was collected from each group at random times and days. After collecting and analyzing several hours of IRC communication, it was found that it was often difficult to associate individuals in IRC with their 'real life' personas in email, making it difficult to use these communications to verify statements made by interviewees. However these communications were valuable both in determining group norms and because they function as a sort of 'back stage' for informal communications.

Finally, all documents and manuals which pertain to the development process, or the functioning of the group were examined (e.g. formal group rules and guides for new developers). These formal documents provided an important insight into group practices, especially because both groups were slow to adopt new rules without a high level of group consensus.

3. Data Collection

In-person and telephone interviews were preferred in interviewing, because electronic communication provides limited access to interactional cues such as expression and tone of voice, which helps to contextualize statements (Markham, 1998). Since the members of the groups under study are geographically dispersed[31], online interviews were conducted for members of the group outside of the United States, unless the interview was considered especially important. Options for conducting online interviews included video conferencing, internet telephony, instant messaging, and chat rooms. The interview

[30] These networks include Dalnet, Efnet, and OpenProjects Net. Generally, large projects consider a few of these channels to be officially sanctioned by the group.

guide (see Appendix) was not strictly adhered to and probe questions were used to follow up on interesting points raised during the interviews. Interviewees were not admonished to "stay on topic," and I found that these unexpected discussions often raise new issues not yet included in current research questions (Glaser and Strauss, 1999).

In addition, interviewees were encouraged to provide detailed examples when possible as support for general statements, for example by "walking through" an organizational process or by recounting specific incidents that provide evidence for generalizations. In the interviews, I adopted the role of "student" who is seeking to learn about the practices of the group, an appropriate role since the practices of the group and the technical issues important to members are often new to me (Weiss, 1994). In-person and telephone interviews were transcribed, and a computer log of on-line interviews was retained for later analysis. A similar computer log was collected from all IRC chat sessions for later coding. Finally, copies of sampled emails were retained and coded. These secondary sources of information often provided valuable insight into interviewee's statements, both by triangulating the validity of their statements and by providing topics of interest for interviews.

4. Validity and Reliability

Multiple sources of information were collected to increase the validity of research conclusions. The primary source of data was interviews conducted with members at various levels of involvement in the groups, including dissatisfied members who left the group. Additional sources of information, including transcripts of IRC sessions, were

[31] For example, about 40% of the members of Star Linux are outside of the United States

used to check the validity of statements made by interviewees. This method of "triangulation" was an important method for reducing bias from the researcher and interviewees, and group members with different roles and levels of status were interviewed to provide a balanced perspective. Interviewees were also asked to provide detailed accounts of incidents and specific examples whenever possible to increase validity, because concrete examples are less likely to be distorted, consciously or not, by interviewees (Weiss, 1994:147).

Verification of data was aided by an attempt by both open source groups studied to retain a high degree of organizational openness. Generally, email lists, which act as the main channel of organizational communication, can be read by anyone who "subscribes" and personal email outside of the email lists is not often used[32]. This allows researchers to monitor communication without interfering, or to "lurk," without creating many of the problem of participant observation such as influencing the actions of individuals observed (Rutter, 2002).

In addition, emails are archived and can be searched by the author of the email and for key phrases. Emails were searched to follow the history of important issues and individuals discovered during study of the groups. Using this method, some of the benefits of a longitudinal study were gained, since it is possible to track a problem or the statements of a specific individual over several years. This helped me to determine whether important issues and problems are short term or are endemic to the organization.

It is important to acknowledge the limits of the sample of the groups selected for study. Operating systems are only one of many types of open source projects. Operating

systems were the first major open source projects and generally have a high profile in the open source community, although they form a minority of open source projects as a whole. For example, SourceForge, the "world's largest open source software development website," only lists 2% of their current projects as "operating systems" and more popular categories include multimedia and web clients, although most of these project only have a few or a single member. In contrast, the Linux and BSD communities have attracted a large number of highly committed volunteers working on various distributions of the software.

Overall there are thousands of Linux and BSD "committers" who are able to make changes without the approval of other members, and many others join the mailing lists to these groups and submit "patches" to committers for inclusion in the kernel. Both Star Linux and LithiumBSD are about the same size with a few hundred members, and both focus on development of operating systems. It is likely that group dynamics and organizational practices will differ somewhat within smaller groups and groups focused on different types of programs, as well as within the dozens of other distributions of Linux and BSD. The Linux/Unix camp has also been around for much longer than other open source projects, and it possible that this may increase the level of formalization within Linux and BSD projects. It is possible that in more established open source groups that past lessons are formalized into explicit rules and organizational structures, and that highly developed systems of normative control have evolved which do not exist in upstart open source projects.

[32] An interviewee in the preliminary sample explained that generally personal communications between developers will be copied to the email list if the issues are pertinent to the group as a whole. There are, however, a few private email lists, which I will attempt to gain access to through the use of an informants.

Since only two groups were sampled this also limits the generalizability of the findings in this study to other open source groups. In order to minimize these limitations, the cases in this study were chosen because they differ in important ways and represent competing camps within the open source community. One important difference is in group age. While the BSD operating system has existed as an open source project in some form since 1984, Linux and the Star Linux distribution did not begin until a decade later[33] and it was found that group dynamics appear to differ within post-bureaucratic groups as groups become more established (See Chapter IV for a discussion of stages of growth for the groups studied).

The two groups also use different licensing schemes, which affects how their software can be used by business and reflects their general philosophy regarding "free software."[34] The two groups are members of competing camps within the open source community and each group has evolved unique work processes, roles and organizational structures. Studying divergent types of "negotiated orders" within open source provided greater insight into the variability between open source groups.

5. Data Analysis

Categories of analysis and theoretical explanations were developed in the process of collecting and processing data in accord with "grounded theory" methodology (Glaser and Strauss, 1999). In "The Discovery of Grounded Theory," Barney Glaser and

[33] However, the specific 'distribution' of LithiumBSD was started at roughly the same time as Star Linux, although some of the original developers became members in the upstart group.
[34] BSD tends to be more business-friendly because they do not require that modifications of their programs be released back to the community, while the GPL license used by Linux requires that commercial modifications to Linux code include source code.

Anselem Strauss propose that samples should be drawn within a theoretical category which has been identified as important during the process of collecting data (such as developers who nosily quit a group only to rejoin a few months or years later) until the point of "saturation" has been reached within that category. Saturation is defined as the point where the researcher can predict with a high degree of accuracy how a member of a theoretical category will respond to her questions.

Grounded Theory is considered to be "theory generating" because patterns of actions, attitudes, and statements observed in interviews and during participant observation define the research questions and theoretical interests (Corbin and Strauss, 1990). These methods are especially well suited to finding new theoretical explanations for phenomena, such as the open source movement, which have not been extensively studied.

The qualitative coding program Atlas.ti was used to code interview transcripts using categories generated during data collection and analysis. Atlas provides several forms of "output" to aid analysis including lists of quotations falling within a specific category. This output was examined to gain insight into the research questions, and to devise new theoretical samples. Transcripts from IRC chat sessions and emails from group listservs were also coded in a separate analysis, using similar categories of analysis. As important theoretical categories developed, email listservs and email archives were sampled in order to observe these issues unfolding in the day-to-day comments of the group.

This "participant observation" was used to provide specific incidents and statements that helped to test issues and research questions raised in interviews. These email lists focus on different topic areas (such as creating software for schools) and different audiences (such as communication between group leaders). Due to the large volume of

communication within many of these lists, which often generate hundreds of email a day, it was important that some type of sample be drawn (Ruhleder, 2000). While a random sampling method could conceivably have been used, in keeping with grounded theory methodology, I determined which emails to read according to their subject headings and authors, after I had read a large sample of emails and learned how to determine whether they would provide insight into research questions. In the early stages of research, at least one email for each subject heading and group member was examined[35]. As the categories of discussion and the members of the group became more familiar, it was be possible to focus sampling on topics of discussion and individuals with greater theoretical interest.

Finally, several sources of secondary data were collected using more informal methods. Documents and manuals on group websites were examined to gain additional insight into group practices and values. Generally, all documents which pertained to the development process and group practices were examined (e.g. manuals for new committers) and compared with the practices of individuals discussed in interviews. Group newsletters and open source 'news websites' were also followed to determine important issues within each group. Along with interview data, this information was used to determine patterns of socialization and status promotion within the group, as well as formal procedures and structures adopted by each group to coordinate work and determine group status.

D. Chapter Conclusions

[35] Often several emails are sent on the same topic as members reply to earlier comments.

Multiple sources of data were collected to gain a broad understanding of the communications and activities that occur within the two open source groups to be sampled. *In toto* these channels of communication form the view of the group of a typical member, because most members participate in the groups using various forms of online communication, such as IRC and email, rather than meeting face-to-face. In addition, organizational archives including email repositories were examined to gain a more longitudinal understanding of the organizations and their members.

Chapter IV: History and Culture of LithiumBSD and Star Linux

> **Principle 1:** *In bureaucracies, consensus of a kind is created through acquiescence to authority, rules or traditions. In the post-bureaucratic form it is created through institutionalized dialogue.*[36]
>
> **Principle 4:** *Because interdependence around strategy is the key integrator, there is a strong emphasis on organizational* mission*. The trend has been to focus on how the company actually seeks to achieve rather than on universalistic statements of values. It is often hard for an outsider to understand what all the effort is about: Most mission statements seem remarkably innocuous. They say something about improving quality and cutting costs, and something about the kind of a business the company is in—rarely anything surprising. But the mission plays a crucial integrating role in an organization that relies less heavily on job definitions and rules. Employees need to understand the key objective in depth in order to coordinate their actions intelligently "on the fly."*[36]

Introduction:

Shared values are the glue that holds together post-bureaucratic groups in the absence of strong leadership and well-defined roles and responsibilities. Gideon Kunda found that the engineering firm that he studied attempted to socialize their workers into a system where they would "do the right thing" technically and internalize the company goals and methods. Morrill found that managers created a system of norms for determining status and resolving conflicts which other managers enforced by punishing individuals who broke these 'rules of engagement,' for example by being to aggressive or 'risk adverse' in their interactions with other managers, by labeling those individuals as too-meek

'pigeons' or overly aggressive 'black hats' who should be avoided if possible (Morril, 1991).

Star Linux and LithiumBSD are highly diverse groups with members from all over the world with very different cultural backgrounds and worldviews, however members of these groups share a high level of consensus that programmers should attempt to "do the right thing" by making decisions based on technical and organizational principles shared by the group. These shared principles, also known as "programming philosophy" will be examined in more detail in the section below.

In this section I will explore the common threads that hold the open source community together and look into the unique histories of the two groups, which have influenced the differences noted by the interviewee quoted in the beginning of the chapter. The first section of this chapter will consider the fundamental tenants of the 'Unix philosophy' which are values that hold together many open source communities. The next two sections will detail the differences between the two groups examined in this study. The final section of this chapter considers a tentative model for the growth of post-bureaucratic groups as observed within the two groups examined within this study.

A. Common Threads of Belief

Despite the existence of dozens of different open source operating systems with different organizational structures and 'group philosophies,' common threads of belief were found within the groups studied. Participants described two common cultural features in Star Linux and LithiumBSD: a commitment to an open design process which

[36] These principles, which introduce most substantive chapters, were developed as an 'ideal type' of the postbureaucratic organization (Heckschler, 1996).

attempts to make programs as simple to debug and read as possible and a commitment to keeping the source code of their programs open to anyone interested in improving it.

Both groups subscribe to a common belief system: the 'Unix philosophy' of programming, which values simple and modular designs (Gancarz, 1995). Modularity is the idea that programs should fulfill a single purpose and be able to be easily used with other programs to which these programs are linked through Unix shell script programming or 'piping.' This programming philosophy allows the user of a Unix system to link the output from a given program, for example a spell checker, with another program, such as a program that formats the text for printing. This allows users of Unix systems a wide range of tools they can easily combine rather than one unitary program that combines many features. The Unix approach disdains programs that attempt to load many functions into a program (e.g. for marketing purposes) while doing none of them 'the right way.' One member of LithiumBSD provides a concise definition of the Unix Philosophy:

> In short, it's to get the best solution for as much of a problem as possible, the 100% solution for 90% of the problem. It's also that all your tools should work together, and rather than have a few very complex tools, it is better to have a lot of simple tools that work together.

In *The Art of Unix Programming,* Eric S. Raymond defines the Unix concept of programming 'simplicity' (Raymond, 2003):

> Questions about simplicity, complexity, and the right size of software arouse a lot of passion in the Unix world. Unix programmers have learned a view of the world in which simplicity is beauty is elegance is good, and in which complexity is ugliness is grotesquery is evil.
>
> Underlying the Unix programmer's passion for simplicity is a pragmatic fact: complexity costs. Complex software is harder to think about, harder to test, harder to debug, and harder to maintain — and above all, harder to learn and use. The costs of complexity, rough as they are during development, bite hardest after deployment. Complexity creates

> places for bugs to nest, from which they will emerge to trouble the world through the entire lifetime of their software.
>
> All kinds of pressures tend to drag programmers into a swamp of complexity nevertheless. . . Feature creep and premature optimization are the two most notorious. Traditionally, Unix programmers push back against these tendencies by proclaiming with religious fervor a rhetoric that condemns all complexity as bad[37].

This ethic of simplicity is evident on the organizational level as well. An important concept in BSD culture is the idea of avoiding debates over 'the color of the bikeshed'. The following quote from a high-profile member of the BSD community explains this notion of 'organizational simplicity':

> [A] good thing about moving decisionmaking down the chain [to the people who implement decisions] is that you can avoid what C. Northcote Parkinson refers to as "bike shed discussions." Parkinson shows how you can go in to the board of directors and get approval for building a billion dollar nuclear power plant, but if you want to build a bike shed you will be tangled up in endless discussions.
>
> Parkinson explains that this is because a power plant is so vast, so expensive and so complicated that people cannot grasp it, and rather than try, they fall back on the assumption that somebody else checked all the details before it got this far. A bike shed on the other hand ... anyone can build one in a weekend. So no matter how well prepared, no matter how reasonable you are with your proposal, somebody will seize the chance to show that he is doing his job, that he is paying attention, that he is *here*.

Members of LithiumBSD argue that excessive 'bikeshedding' within the project creates unnecessary rules that hinder the project.

An additional important principle uniting the open source community is the idea that whenever possible, source code should be available to any interested parties to review and modify. This allows programmers to 'build on the shoulders of giants' by developing new programs using well-tested and peer-reviewed software solutions. While it may appear that these groups are united primarily by shared technical principles, this 'ethic of

[37] Raymond, 2003. See http://www.catb.org/~esr/writings/taoup/html/ch13s01.html#id2964982 for more detail on what he means by complexity

openness' also extends to the organizational structure of the groups as a whole[38]. One effect of opening source code is that it allows for easier peer review or as 'Linus's Law' states "with enough eyes, all bugs are shallow" (Raymond, 2002).

One major difference between the BSD and Linux camps is their willingness to protect the 'openness' of their source code through legal means. For example, companies can integrate BSD source code into a product and resell the product as a 'closed source' product. In contrast, Linux uses the GPL licensing scheme, which some critics label as a 'viral license' because software that incorporates Linux source code typically must also be released as open source or be considered in violation of the copyright. It is possible that the greater participation by programmers in Linux projects such as Star Linux when compared to BSD projects is explained in part by this legalized protection of contributions which ensures that any changes to their code will remain open to the programming community. In contrast, contributors to BSD projects hope that their more permissive license will encourage the standardization of their operating system among business. They have found a great deal of success in this area as BSD is responsible for supporting much of the infrastructure that runs the Internet and email servers.

B. LithiumBSD: A Guild of UNIX Gurus

In many ways the BSD system has provided the infrastructure for the Internet, including the TCP/IP communication protocol which makes Internet communication possible and sendmail, which processes emails as they travel through the Internet. Despite the importance of the BSD system to computing history, BSD has not gained

[38] See Adler and Borys, 1996 for evidence of the relationship between task and organization openness in

much of a market share in the computing world, due in part to its unique licensing agreement which allows corporations to use BSD source code without requiring that any improvements they make be contributed back to BSD[39]. A few originators of the BSD system have become influential figures in the computing world including Bill Joy, the founder of Sun Microsystems.

The BSD project was originally funded by AT&T in 1984 and was developed by the University of California to replace their archaic mainframe system. After legal issues with AT&T over ownership of the source code, several developers in the project broke away from the original project to rewrite the proprietary portions of BSD from scratch, to avoid copyright issues. Today these efforts have been continued through three 'distributions' of BSD. Distributions are teams of developers who have a unique view of how the BSD system should develop and choose to develop the operating system independently from competing distributions, although the groups may share source code between projects. Two 'distributions' of BSD developed soon after the AT&T lawsuit was settled: FreeBSD which emphasizes making good fileservers (e.g. Internet and email servers) and NetBSD, with a strong emphasis on portability, which is the ability to run on a large number of systems such as Macs, PCs and Solarius systems. The slogan of the latter group is ``Of course it runs NetBSD,'' and the group claims, tongue-in-cheek, that one day NetBSD will run on any computer system or electronic device including toasters. More recently a 'fork' of the FreeBSD project arose after a contentious and ongoing dispute in 1995 with a leading developer, Theo de Raat, after he had ongoing personal

---

the workplace.

[39] In contrast, Linux is licensed under the GPL (General Public License) which requires that any changes to open source code be open source as well, although the strength of the GPL has yet to be fully tested in court).

and technical disagreements with several senior members of FreeBSD. De Raat started OpenBSD with a group of FreeBSD developers who supported his views, and they are attempting to create a distribution of BSD that emphasizes security over portability. The remainder of this section will consider the history and culture of one of the BSD distributions, 'LithiumBSD.'[40]

While few of the original members who created the original BSD system remain with LithiumBSD, many members pride themselves on being part of computing history. Several key members of the group have belonged to the group for many years. One long time member of the group who recently resigned offered this assessment of the group:

> It's a family organization. It's not a group that has a lot of turnover. The core engineers have been there for about 10 years and they've known each other for about 20 years and they're still a tight family . . . They get very high level technical people.

While there is a great deal of diversity among members, who include university students, independent consultants and programmers for major software companies, several members described a unifying philosophy of the group as being 'doing things right':

> Part of being a manager is making compromises to get the job done, and part of what fuels people to work on something like LithiumBSD is they say 'I'm going to do it right, I'm not going to do what's fast, I'm not going to do what's expedient, I'm going to do it right.' Maybe I'm nuts but this is what I think is right. I want to do it and I want to prove that it's the right thing to do. There's sort of an aversion in LithiumBSD to too much management, too much meddling. You don't want to make compromises; you want to implement the best technical solution, even if that requires fighting it out.

[40] A pseudonym

Another member explains that "it always comes down to 'doing things right,'" although he acknowledges that this motto is "a decidedly vague phrase, and so there can be some confusion about what is 'appropriate' for LithiumBSD."

The majority of members of the group believe that LithiumBSD functions as a meritocracy by allowing the best solutions and most skilled members to rise to the top through the process of peer review. A few past and current members of the group disagree: an ex-member of LithiumBSD who held a central role in the group and resigned after he didn't get significant recognition for his contributions to the group described LithiumBSD as an 'elitist clique' which isn't open to new approaches, rather than meritocracy. This outspoken critic of the group believes this is why Linux has gained such a large gain in membership in the past few years, despite the fact that BSD projects have been around longer. He explains:

> LithiumBSD is not really attracting new developers from Linux distributions because of their inability to receive new patches [which are changes to the source code to fix bugs or add new features] and you don't see positive feedback in patches as you do in Linux. You see very radical changes to the Linux kernel because Linus allows for a meritocritous process on mailing lists. He allows for technical arguments to be explored and the critical thing is that he isn't an either/or person.
>
> He gives you the opportunity to experience with potential solutions and the popularity of new techniques rattles the old schoolers. . . unlike the BSD's who have to make changes almost by committee. And changes have to conform not just to the committee itself but to the sensibility of the committee, what they think is good as opposed to what is good. They really can't afford that because they don't have the experience that the Linux people have with that kind of thing. The Linux people in a mad rush have developed the skills necessary to deal with all of those problems.

In contrast, another developer with a view more typical of current members identifies LithiumBSD as a group of 'Unix gurus' who are committed to the group

despite the 'hype' that has been generated around Linux. He provided the following comment when asked to identify a 'unique LithiumBSD culture'[41]:

> I wouldn't call it unique. Open source in general often has a "we do it because we like it" philosophy. While Linux might have lost some of it and become a "hyped" operating system that you run to be cool, BSD is still very much a project run by volunteers that love what they do and strive for technical excellence, not fame and fortune.

C. Star Linux: A 'Stepping Stone' for New Programmers:

Linux distributions are collections of various free and 'non-free' programs, which are modified to work with the Linux kernel. Unlike BSD systems, members of these groups almost never edit the Linux kernel itself. Rather, they modify various open source and proprietary programs (such as data analysis software) which are 'ported' to work with the kernel created by the Linux project headed by Linus Torvalds. As a result, the Star Linux group has a lower barrier to entry than LithiumBSD group in terms of the level of programming skill required to participate in the group. Many participants in Star Linux joined the project to learn how to widen their programming skills. As one member explains when asked whether his participation increased his technical skills "definitely, tons. It's how I learned what I know about computers."

Culturally, Star Linux tends to be more inclusive than LithiumBSD. Members can contribute directly to the project with little or no supervision from other members, however to become a formal member of the group, there is an extensive process of initiation[42]. As a result, many members of Star Linux gained their first practical

[41] The culture of the LithiumBSD group will be further examined in future interviews which focus on how sub-groups in the each project negotiate tasks and roles. See Section III for more detail.
[42] See the section on Role Negotiation for more information on this process

programming experience when they started contributing to the project. The ease of contributing to this project combined with its high profile within the free software community have drawn a large number of new developers to the project, many of whom might never have become involved in programming otherwise.

A final important element of the Star Linux culture is the group's passionate defense of free software principles. Unlike the BSD license, which allows any user to use and modify the source code created by the group in any manner they wish, the GPL or 'copyleft' requires that any programs that use open source code also be released as open source. The passionate defense of the 'freedom' of source code made by members of this group has probably been a major factor in encouraging new programmers to join their project, as new members can be sure that their contributions to the group won't be adopted by companies who 'close the source' and profit from their efforts without releasing improvements to the code back to the programming community. As the next two chapters demonstrate, this 'ethic of openness' applies not only to source code, but is also an important principle in creating work processes as well as determining the rules of the organization itself. LithiumBSD and Star Linux both manifest the postbureaucratic principles listed at the start of this chapter. These groups evince both a strong commitment to their group 'mission' which includes an 'ethic of openness' and a reliance on institutionalized dialogue in determining work processes (See Chapter 5) and in determining group roles and settling conflicts (See Chapter 6).

Principle 6: *The focus on mission must be supplemented by guidelines for action: these, however, take the form of principles rather than rules. The difference is that principles are more abstract, expressing the reasons behind the rules that are typical of bureaucracy. The use of principles carries a major advantage and a major danger. The advantage is that principles allow for flexibility and intelligent response to changing circumstances: People are asked to think about the reasons for constructing on their actions, rather than rigidly following procedures. The danger is that this flexibility can be intentionally or unintentionally abused, threatening the integration of the system. The dangers are reduced by two mechanisms: the creation of trust, derived especially from a clear understanding of the interdependence among all; and by periodic review and discussions of the principles to be sure that they accurately capture what is needed and have not been distorted. Post-bureaucratic organizations spend a great deal of time developing and reviewing principles of action.*

Principle 7: *Because of the fluidity of influence relations by comparison to offices and authority, decision-making processes must be frequently reconstructed; they cannot be directly "read" from an organization chart. The choice of "who to go to: is determined by the nature of the problem, not by the positions of those initially raising it. Thus, processes are needed for* deciding how to decide*—what might be called "meta-decision-making" mechanisms. In a number of companies there are cross-functional and perhaps cross-level committees that sort issues and try to develop appropriate processes for each of them. To choose a single example: At a Shell plant in Canada, issues that cannot be dealt with by individual teams of workers go to a "team norm review board" composed of operators, union officials, and managers; this committee then establishes a way of resolving the problem—they may, for example, set up a subcommittee, or a series of meetings of the affected groups, or a call for additional information from inside or outside, or some combination of these tactics.*

How is work divided up within Star Linux? Who, if anyone, makes sure that important work gets done? That's one of our weak spots, in my opinion. There isn't really any way to insure that something gets done except to do it yourself. Chances are, if the work is important enough, somebody will take the initiative. If nobody takes the initiative, then it couldn't have been that important, could it. ;^)

--Star Linux member

Introduction

Many economists and organizational sociologists have noted the rise of work teams and 'virtual organizations,' especially within companies engaged in intellectual labor such as software programming (Castelles, 1997, Drucker, 1998). For example, a 1999 survey by the University of North Texas Center for the Study of Work Teams (CSWT) found that by 2000, 80 percent of Fortune 500 companies predicted that they would have over half of their employees on 'work teams' (Cited in Joinson, 1999). Similarly, some organizational scholars have noted the rise of a new, less coercive style of management, which allows workers a great deal of autonomy in monitoring group rules and deciding how to solve work tasks. Stephen Barely and Gideon Kunda note that American industry has cycled through different forms of managerial control, moving from coercive to rational to normative control. Normative or 'concertive' control uses employee internalization of group principles rather than rational self interest to encourage employee behavior that benefits the firm. Barley and Kunda take a pessimistic view of these developments noting that "by winning the hearts and minds of the workforce, managers could achieve the most subtle of all forms of control: moral authority" (Barley and Kunda, 1992; See also Barker, 1993). In contrast, other observers of these trends believe that the increased trust and autonomy that workers gain within post-bureaucratic organizations both increase worker motivation and decrease levels of 'worker alienation' (Lawler, 1996). Whatever their effects may be, this increase in the prevalence of work teams and virtual organizations appears to be an adaptation to market trends as the increasing growth of information technology and globalization pushes industrialized countries towards 'network societies' (Castelles, 1997).

According to neo-contingency theorists, while organizations that engage in relatively stable or 'mechanical' work processes function best under traditional command and control systems of management where work practices are codified (Galbraith, 1977; Van de Ven and Delbecq, 1976), more complex or 'organic' labor is best managed by the workers themselves, especially if the industry requires organizations to rapidly adapt to changing circumstances. Less hierarchical post-bureaucratic organizations are seen as a good 'fit' for an unstable market because they are able to react more rapidly to market uncertainty by allowing employees to self manage their contributions to the organization and to police the behaviors and attitudes of their 'teammates' (Ouchi, 1980).

In their article "Occupational Communities: Culture and Social Control in Organizations," Van Maanen and Barley develop a typology of trust-based organizations which is useful for understanding typical methods for coordinating post-bureaucratic organizations in the absence of formal bureaucratic systems (Van Maanen and Barley, 1984). According to Van Maanen and Barley, trust-based groups are primarily coordinated through "rapid feedback reward systems" that are in agreement with group culture. Control systems of this type include peer review of employee work standards. Concertive control acts upon individuals in trust-based organizations through the mechanism of high status members awarding or denying status to other members based on their conformance with group culture. These high status members, which include group leaders and managers, are then to control the group by encouraging other members "often symbolically" to pursue "certain goals, attitudes and behaviors" (Van Maanen and

Barley: 175). Lastly, they explain that these standards are upheld by enforcing rigorous selection and orientation processes[43].

As the following sections of this chapter will show, LithiumBSD and Star Linux rely both on autonomous experts who are able to apply general group rules to new situations and on formal coordination systems developed by members to enforce the 'best practices' defined by the group. The following section of this chapter explains how Star Linux and LithiumBSD conform to many of these post-bureaucratic behaviors as would be a good 'fit' for organizations that engage in complex and highly-coupled work—the programming of highly complex operating systems.

A. Bug Tracking and Coordination[44]

Both LithiumBSD and Star Linux are highly complex projects, which rely on highly developed systems of coordination. Complex coordination systems are used in both groups to track various types of bugs and to assign them to appropriate individuals whenever possible. Peer review and bug fixing are organized through several systems, including specialized mailing lists to deal with testing of source code and security. Due to the large amount and variety of source code required to build an operating system, members rarely read source code that falls outside of their area of expertise. Rather, the primary system of peer review is 'problem reports' from users and other members of the group. These bug reports are sent through a tracking system, which logs the severity of

[43] These processes of role negotiation will be detailed in Chapter VI on Role Negotiation and Conflict Resolution.

[44] A quick summary of the architecture of an operating system: Operating systems including Windows, Linux and BSD consist of a 'kernel', the core of the operating system which provides low-level instructions

bugs, and the time that has passed since they were filed. As a member of Star Linux explains: "I think a huge amount of the actual work in Star Linux is triggered by developers filing bugs on other developers' stuff."

A major coordination challenge which is managed by formal and informal coordination systems is tracking the changes made to the source code by individual developers. The main method for coordinating programming efforts within LithiumBSD is for developers to make their intended changes clear on the appropriate mailing list to give other developers fair warning that the changes will be made. Since developers are all working on the same 'kernel' for the operating system, there is always a possibility that changes that one developer makes will adversely affect other parts of the system and that developers may be pursing competing efforts or editing the same source code. The most difficult efforts to coordinate are large subprojects which 'touch many parts of the system', requiring the participation of group members with a wide variety of specialty areas within the project. In LithiumBSD, these project-wide efforts are publicized through monthly status reports which communicate to the developer community any major changes in the works.

Another major coordination problem that arises within these groups is the coordination of multiple members who may be simultaneously making changes to the same area of the source code. As a long-time member of LithiumBSD explains, this can give rise to 'patch time conflicts':

> So what a member does is take a snapshot [saved copy] of the current version of LithiumBSD source code to his machine and he makes all of his changes which again may be many thousands of changes . . . Ok, now you're ready to add these changes back into the main source tree [where

---

for many important processes, as well as additional programs (e.g. Microsoft Word) which allow the user additional functionality.

> the main version of the source code is stored]. Well obviously all that work isn't going to happen in zero time and you don't know who else is taking snapshot [by saving a local copy of the source code for editing] and working at the same time. So basically you have the notion of "patch time conflicts" where nobody objects to the code but you're changing the exact same line I'm changing for a different reason.
>
> Both sets of changes may be good and everybody may be happy with them, except you can't just apply the patch because the tools have no idea how to merge these two changes. And that's where you just do the best you can. When there's no philosophical disagreements, or coding disagreements or technical disagreements, you just try to pay attention to who else might be working in this section of code and you try to give people what's called a 'heads up' and you literally send a message with 'heads up' in the subject. You say "hey I'm going to be working over there, if that bothers anybody or if anybody else is about to do something, you'd better say something now because one week from now if you interrupt me I'll be really irritated. If you interrupt me now I don't care because I have nothing at stake. I haven't typed anything yet. But a week from now if you commit something that causes me to rewrite my work, well then I'm not going to be happy."

As this example illustrates, mailing lists are an important channel of communication for both groups. Both groups have a few central mailing lists that the majority of members subscribe to and a many lists which are focused on specific topics including subprojects within the group or functional divisions, such as members who are interested in improving the security of the operating system. Members explained that this specialization is necessary because it's easy to become overwhelmed with information otherwise. Both groups have dozens of mailing lists each with anywhere from one email a week to several hundred. A long-time member of LithiumBSD explains that "as a developer, you *have* to isolate yourself from at least some of the chattering that is going on, because no one could possibly read all the LithiumBSD-related email which is posted on a single day."

Many developers in both groups also use IRC (Internet Relay Chat), especially for quick answers about specific technical issues. It is not uncommon for a member to

remain connected to an IRC channel all day even if they are not around, and only later skim the discussion for interesting topics of conversation. IRC discussions also tend to be more personalized and conversational and a useful communication channel for small problems that developers need immediate advice to solve. As one member of Star Linux explains: “Real things happen on the mailing list. IRC is sometimes just a first stage [for developing new projects] or a backroom or whatever.”

Within LithiumBSD, there are also formal and informal systems to allow members to temporarily claim ‘ownership’ over areas of the source code in order to better coordinate programming efforts. The primary formal method of claiming ‘area ownership’ involves the listing of a member’s name on the LithiumBSD website, indicating that he is an expert in programming a given area of the operating system. This listing generally suggests, or much more rarely explicitly requires, that other members clear changes within the area that they ‘own’ before committing them to the source code.

Without this system of organization it would be possible that different members of the group working on the same project might make conflicting changes to the source code. This system of ‘area ownership’ also ensures that there is one individual who is ultimately responsible for bugs within a given program, although this ‘ownership’ can be superseded by other developers when needed. A long-time member of LithiumBSD explained the system of ownership in the following way:

> When a developer does a commit, there is a template of information which pops up with various fields the developer can choose to fill in. One of them is "Reviewed by". I say "Reviewed by: PaulW", and there's pretty much no one who will complain. Between me and him, there isn't anyone in LithiumBSD who knows anything about networking that we don't know. . . So, in some sense that's hard, and in some sense it's easy. It's hard because Vance always has good ideas which I feel obliged to include, so the review process takes awhile, maybe a week or two. But once I get

> past him, it's easy sailing. For other things I change, I have to search around for someone who might care, and yet you can't just keep flooding the mailing lists saying "So, who cares about XXX?". That just annoys people more. So, you check the commit-history for whatever it is you're looking at, and you send a message to whoever has worked on it recently.

The Star Linux project relies on a similar system of 'ownership' but with a distinctly different purpose in mind: ensuring that at least one member is responsible for the thousands of programs that are 'ported' into the operating system. Bugs in Star Linux are filed against specific programs or 'packages' in the operating system, each of which is 'owned' by a single person who is responsible for fixing problems within the program. Bugs may include incompatibilities with other programs, or system-wide problems (e.g. a bug in a core package that slows down the operating system as a whole). Star Linux generally uses a systems of "one owner per program", except in cases where the program is especially complex, in which case 'co-owners' share responsibility for the program[45]. Becoming a 'package owner' for dozens or even hundreds of packages is highly encouraged by leaders and others as there generally are many packages which are 'unowned' and have no member personally responsible for fixing bugs. In order to become a member of the project, prospective members must adopt at least one 'unowned' package and demonstrate to a current member that they are able to fix bugs within the package.

In contrast, many members of LithiumBSD have expressed reservations regarding the system of 'area ownership' over portions of the source code. These critics of the LithiumBSD ownership system do not respect members who are too assertive in forcing other members to clear changes through them, especially if the 'owner' has not

[45] A 'program is an external program such as an open source word processing program which is adapted by the 'owner', in this case the prospective member, to function within the Star Linux operating system.

demonstrated sufficient technical expertise in the area he 'owns.' Rather, informal processes of coordinating work are preferred by most members. As a member of LithiumBSD explains:

> I think that the people who do feel more threatened and they feel that they need to assert maintainership. Even more fundamentally then that, it's not a technical thing, it's a personality thing where you have a few people who can't work with each other.

While both of these systems of ownership may appear to be typical bureaucratic divisions of labor, these systems of tracking responsibilities within the groups have little practical effect without informal mechanisms to encourage their use. The remainder of this chapter focuses on these informal systems of control that supplement the formal tracking systems in both groups. These different systems are briefly outlined in the following diagram and then explained in detail below.

DIAGRAM 1: FORMAL AND INFORMAL COORDINATION SYSTEMS

	Star Linux	**LithiumBSD**
Formal Systems	'Package Ownership' (Required) & Bug Tracking System	'Area Ownership' (Fairly Rare), Bug Tracking System, & 'Heads Up' Messages
Informal/Supplementary Systems	'Bug Kills,' 'Changes by Non-Owners' (CNOs), 'Cheerleading'	'Cheerleading,' 'Posting Temporary Fixes,' Delayed Changes, Ignoring 'Area Ownership' (Can lead to 'commit wars)[46]'

[46] See the Chapter on Role Negotiation and Conflict Resolution for definition

Both groups strongly rely on informal mechanisms for tracking bugs in projects, in part because it is impossible to fully automate the discovery and assignment of bugs and in part because members of the group can always refuse to fix problems that are assigned to them. Members use a formal automated system (similar to a compiler) which discovers basic errors that will prevent the program from working properly with other programs or the operating system, although more complex bugs are much harder to find. One informal system for resolving these problems relies strongly on the self-assignment of tasks by members: 'bug kills' organized by members of the Star Linux project when there are a high number of severe or 'release critical' bugs. During these online gatherings (which usually convene on IRC), interested members and the public communicate in detail about the best way to resolve important bugs as well as who will be responsible for implementing the solution. These online gatherings function as a sort of 'rapid peer review' where members are able to quickly share technical expertise while focused on specific problems. Bug kills help the group to focus on important problems that might otherwise be buried within the deluge of email related to bugs in the software. A senior member of Star Linux explains how bug kills are organized by the most committed members who often elicit the participation of new members in the group:

> The organizers are usually Project Quality[47] people who want to speed up the release process: Vaste and I have both done it in the past, along with Jordi Mallach and others. Developers participate for much the same reason, and it provides a convenient framework for lots of related things to be done at once, which is a useful economy of scale when you're talking about several hundred release-critical bugs. Also, it's a good way to help new developers get involved in work beyond their own packages (and lets

[47] A subgroup responsible for ensuring that the operating system is up to par before new versions are released.

> us do a bit of mentoring). CNOs have been known to make things worse, yes. It usually leads to a conflict when they do.

Despite a proliferation of formal tracking systems in both groups, these formal systems generally act as a backup for the informal systems that coordinate most group activities. Within complex projects such as LithiumBSD and Star Linux, coordination of tasks by members of the groups is essential. For example, due to the high complexity of the LithiumBSD system, it is not often clear who is responsible for fixing specific bugs, often because bugs may appear in several parts of the system. The process of assigning these bugs provides an important example of self-assignment of tasks, as one member of the group explained when asked how bugs were assigned in the group:

> A person finds out one way or another if a bug falls into their interests. If someone is known to be good with something (as I mentioned previously about the division of work) they might be told by someone, or have a bug assigned to them. In general, I think, people find bugs themselves [rather than having bugs assigned to them].

Another important informal system used by Star Linux to informally organize bug fixing is the use of "Changes by Non-Owners" (CNOs), or changes to a program which are made by a developer, or a group of developers, who is not responsible for the program[48]. Since any member of the group can make changes to any part of the source code at any time, it is possible for developers to make changes to programs that they do not 'own'. However, if this is not done diplomatically, it can create conflicts between members of Star Linux. To minimize these conflicts, the group has developed this informal protocol, which involves allowing the current owner a reasonable period of time to make the changes himself before a 'non-owner' overrides the owner's responsibility and fixes the bug himself. Interviewees also emphasized the importance of other

members and non-member users who reported bugs in programs that they would have difficulty locating themselves.

In addition to tracking the many changes that occur within each project, both Star Linux and LithiumBSD have instituted several formal and informal mechanisms of peer review of source code. A primary form of peer review in LithiumBSD is review of changes made to different parts of the kernel by interested experts in a given area of the system, and this ensures that most changes get some level of expert review. In addition, the group maintains a list of areas where developers who are experts for a given area can require that all changes be reviewed by them first, or more commonly, that the changes don't 'break' the system.

Both groups also recognize the importance of encouraging other members to become involved in fixing bugs or helping with new subprojects in the groups, since tasks cannot be formally assigned. A member of Star Linux provided the following comment when asked to describe the process of reviewing the source code for bugs:

> At root, the mechanisms for ensuring fixes are social. The TS (tracking system for bugs) itself is really just there to ensure that we never forget about anything and that people can get a convenient list of what they're supposed to be doing. Normally, we have to trust that maintainers are sane and have a good grasp of the packages they maintain. In general, this is a good assumption and keeps people motivated, but there do have to be safety catches.

One danger of trusting other members to police the quality of their own work and the work of other members is that they will not do so due to a lack of time or interest. The same member of Star Linux explains this problem:

> The most common failure mode is that maintainers just disappear, or perhaps get overloaded. So we do have lots of bugs that have been sitting around for some years. We have a "changes by non-owners" mechanism

[48] See the section on Conflict and Role Negotiation in this chapter for more detail on this process.

> whereby somebody else can step in: it works, although it tends to be the cause of arguments if used hastily.
>
> If fixes are bad, we do ultimately trust in the community (both developers and users) to notice; and usually they do. There's no formal commit review process to actually check over source code changes, though.

Three primary methods used in both groups to informally coordinate and motivate work are detailed below: 'cheerleading,' posting temporary fixes, and the self-assignment of responsibilities. Self-assignment of tasks is very common, and it is rare that developers be removed from a task or position for a lack of technical skills or making an error, although informal systems are occasionally used to shame developers who are deemed not up to technical par or who are shirking their responsibilities. Responsibilities and many formal roles in the group are not only self-assigned but can be dropped at any time if the members lack time or are dissatisfied with the direction that the group is taking. To keep track of tasks no one is currently in charge of, both groups have developed multiple systems to track 'orphaned' projects that new members of the group, or established members with extra time, can take responsibility for. If a developer no longer has time to carry out her responsibilities, she can simply "orphan" them, allowing other developers to easily take them over.

Other methods that have evolved in both groups to gain the interest and involvement of other members are 'cheerleading' and 'posting temporary fixes'. The latter method was explained by a long-time member of LithiumBSD as follows:

> Unless you are paying someone, you can't do much of anything to officially delegate tasks. All you can do is cheerleading, or simply debating some ideas until the other person gets excited about it. Another tactic which sometimes works is for you to commit *some* simple change which is "okay", but which another developer is convinced is the wrong direction to take. Make it clear that you're just doing the change "because

> this is the easy short-term fix", and the other guy will appreciate your position, but they'll then get fired up to implement The 'Clearly Superior Solution'.

'Cheerleading' is a method for motivating other members to help solve a problem or participate in a project that is considered important by the 'cheerleader'. This method involves getting other developers interested the problem that they want to solve, generally by posting emails to group lists which advertise the problem and offer possible solutions. While it may require more effort in the short run to motivate other developers through 'cheerleading' and other tactics than to rely on traditional organization systems such as delegation, one member explains that he thinks that in the long run, this tends to lead to better written code:

> In Star Linux the advantage is people's interest in what they do. Given that people only do what they find interesting (and are capable of doing) most jobs are done extremely well.

In order for these informal mechanisms to function as systems of work coordination, it is important that there be a high level of trust within the groups so that they can function as effective workgroups[49]. In the past, members of both groups generally knew and trusted each other, in part because at the start of these projects most people on the Internet were affiliated with universities and other institutions. Both groups have attempted to develop systems that retain a level of trust within the group, including a requirement that new developers meet a current developer in person, which not only "verifies their PGP key[50]" but also encourages some level of social interaction among members. Similarly, attempts have been made to create a "web of trust" where members vouch that other members can be trusted, as well as a required period of mentorship,

[49] See dissertation proposal for literature on post-bureaucratic organizations and the effectiveness of the 'clan' model of organization in complex and intellectual labor.

which ensures that new members are socialized into group practices, and that they share the values and principles of the group. It appears that as long as these groups are able to maintain this level of trust despite their growth in membership, they will maintain 'clan' model of organization.

A lengthy period of socialization and a high level of in-group trust are required to form these highly-cohesive work groups. Once formed, these 'clans' are able to function with minimal formal supervision because group members are able to reinterpret group principles to fit new situations and tasks. The process of creating the formal and informal rules which govern these two groups will be considered in the next chapter[51].

B. Creation of Group Practices and Policies

A member of Star Linux with a central role in developing group policy explains that the formal process for changing group policies[52] is "trivial" and simply involves "convincing three people to support the change and sending it to the Project Administrator, and then you can call it to a vote." However, he explains that in practice the process of changing policy is much more time consuming:

> The only caveat is that in order to get it passed you need to convince enough people to vote in your favor. [In one recent case] we spent about six months writing the drafts and now we have a draft that everyone agrees on. Now we just need to send out a general message and say 'this is a Developer Proposal, vote on it' and five other people co-sponsor the change and I have to start counting votes. What's keeping it at this point is that the people that are pursuing it are busy and as soon as I have time and I know that I'm ready and able, I'll push for a vote.

---

[50] This involves verifying the identity of a prospective member by meeting them in person.
[51] See the Chapter on Role Negotiation and Conflict Resolution for more detail on the New Member process
[52] There are many group policies which cover everything from rules regarding communication with other developers and the proper rules for uploading changes to the project repository.

The result of a slow adoption of new policy changes in LithiumBSD and Star Linux and a reluctance to force members to obey formal rules is that group policy tends to reflect the 'best practices' of the group. Members of both groups tend to have a high level of agreement with group rules, in part because proposed changes to group rules have been debated for months or years before they are accepted as group rules. As one member of Star Linux involved in documenting group policy explains, there is a strong emphasis on consensus in determining policy:

> As a policy editor, my general duty is to maintain the Policy manual and the related sub-policy documents [which provide rules for subprojects in the group]. This means watching the discussions on the policy mailing list, shaping up patches to the document, making sure the formal process for policy changes is followed.

Many of the rules that are created tend to be general principles rather than formal policy, although there are also detailed technical rules for making changes to the source code. This model appears to conform with the 'clan' model of organization described by William Ouchi, where internalization of abstract principles which are then applied to specific and often complex work problems is seen to be highly efficient in coordinating complex labor such as computer programming. An example of one of these principles that was generated through the Star Linux policy creation process is the following general suggestion for member to follow when adding a new program to the operating system:

> The program description [which allows users to differentiate among literally thousands of open source programs they could possibly install into the operating system] should be written for the average likely user, the average person who will use and benefit from the program. For instance, development programs are for developers, and can be technical in their language. More general-purpose applications, such as editors, should be written for a less technical user.

Developers are given a wide degree of latitude in defining their work process as long as their finished code does not create problems for the rest of the system. Generally, developers can use any approach to solve a problem, as long as the finished result achieves the specified goal. For example, while most software development firms define a formal policy for commenting their code (adding text to explain the program to other developers), both groups use only voluntary standards. As a member of Star Linux explained "we don't tend to document coding style . . . it's too contentious."

A common ethic in both groups is a strong resistance to any rules or practices that slow down development efforts (referred to as "bike shedding" in LithiumBSD[53]). A long time member of Star Linux provides an analogy to explain that while the group has adopted some degree of policy and formal processes, it is a necessary evil:

> We should be looking at Star Linux from the point of view as a bunch of guys going fishing. It started out as a couple of guys going fishing, it was informal and as more and more people started going fishing we had to set down dates and now half of the town is coming with us. But in no way have we reached the point where we are a trawling company and need to have boats, and lawyers and insurance, rules and regulations and captains and so on.

And a developer in the LithiumBSD project explained that a dislike of the type of 'unnecessary bureaucracy' he experience in his professional career is common among most members of the project:

> A lot of the people who are working through LithiumBSD are adverse or are not comfortable with too much structure because they've been in commercial jobs or universities or whatever where so much time was spent organizing and deciding. Here at TLU, for a while we just went meeting happy and you couldn't change the text of a log in message without a meeting and getting everybody to sign off on it. And basically what happened is that nothing got done because all you could do was to go to meetings. And sooner or later somebody would miss something anyway

[53] See Chapter IV or glossary for definition.

> and be mad when it happened. So you could never solve the problem and endlessly you would grind to an absolute halt.

Similarly, a developer in Star Linux explains that he joined the project because of its ability to meet project goals despite its non-hierarchical structure:

> Star Linux is one of the few Linux distributions which is so flat – anyone can join and help and get as far in the group as they have the time and intelligence to contribute unlike at work where you can wander off and slack off for months and then come back. I've done that a few times :) Not only is Star Linux flat but it's pretty well organized.

And another Star Linux member explains that most projects are initiated by developers rather than leaders:

> There are all of these mini-projects that start off in Star Linux and most of the time there not started because the project leader say okay, we need work in this area .. . like Star Linux for Kids started when a subset of people decided that they would have a set of packages that would be appropriate for their kid. So decisions aren't made from the top down as you would expect in a regular company. Decisions are made and work is started when people feel that they need to have it done.

Despite comments by several members disvaluing bureaucratic rules as an unnecessary hassle, both groups have developed an increasing number of rules as they have grown and as their members are less likely to know each other directly. This shift can be seen in the following quote by a member of Star Linux who described the evolution of a formal process for accepting new members into the project:

> I started at Star Linux back when Erin Django was project leader so it was a lot different. I was a user with suggestions for changes in a specific program. Since no one was responsible for those changes, he told me "okay, make those changes yourself" and a few days later I was given a log in name. Today it's a lot more complicated, in part because the group has increased so much in size. There is an official Application Management process and unfortunately there can be politics in this process where people can be stuck for months at different steps, complaining no one is doing anything about their application.

At first glance, it appears that the high level of formality within both groups, including formal policy documents and well-developed coordination and tracking systems, belies an avowed dissatisfaction with excessive bureaucracy. However in practice, many developers have only a passing familiarity with the formal policies of the project they contribute to, in part because many of the formal rules have little bearing on their day-to-day contributions to the project. Rather, they typically communicate with other members to determine the best solutions to problems or follow the rules they learned when working under a mentor.

This model of work coordination appears to follow the 'concertive control' model typical of clans (Ouchi, 1980) where workers police their own work practices and the activities of their 'team.' In the following quote a senior member of Star Linux explains that he is very aware that a group culture of peer review motivates high quality work and reinforces high programming standards (italics mine):

> There is no practical way to control the work of an individual as it is being done. By necessity all controls are either before the work or after the work. Also the *pre-work controls are weak by nature*, there is no way of preventing anybody doing any work in any way they want to do it, the only real control is when incorporating that into Star Linux. On the other hand, *even weak controls are very effective on the long run, since the main rewards are peer recognition, which cannot be achieved with poor quality work, and work satisfaction, ditto.* This means that one of the most important issues is induction of new developers. . .
>
> There are almost no enforced reviews or strictly enforced check points, but there is a *very strong culture of peer reviews.* Even though change management and configuration control are rather informal, I still consider the achieved result to be at least as good as in many commercial organizations. This is mainly due to very strong commitment to quality in Star Linux and community effort in doing the work.

This chapter has shown the norms and structures that have evolved within Star Linux and LithiumBSD to assign and coordinate work. These principles for action and

processes for coordinating work fit ‘Principle 6 and 7’ of the postbureaucratic type cited at the beginning of this chapter. A great deal of time is spent in each group deliberating over rule creation, as well as the process of creating rules which Charles Heckscher refers to as “meta-decisionmaking mechanisms” (Heckscher: 223). Members generally follow these voluntary rules because they understand and support their purpose. Rather than enforcing these rules formally, the group relies on informal systems such as ‘cheerleading’ and ‘implementing partial solutions to problems’ to encourage (or shame) other members to fulfill their organizational duties. As the next chapter will show, new members are only accepted into the groups when they can demonstrate that they understand and can apply the technical and organizational principles that provide the foundation for the group. The process of role negotiation and conflict resolution that evolved within these groups to fill the lack of formal systems of task and role assignment are also examined in this next chapter.

Chapter VI: Role Negotiation and Task Assignment

> **Principle 2**: *Dialogue is defined by the use of influence rather than power: That is, people affect decisions based on their ability to persuade rather than their ability to command. The ability to persuade is based on a number of factors, including knowledge of the issue, commitment to shared goals, and proven past effectiveness. It is not, however, based significantly on official position. Relations of influence can and do form a hierarchy: some people are more persuasive than others. Thus this system is not in any strict sense "egalitarian." But the influence hierarchy is not embedded in permanent offices, and is to a far greater degree than bureaucracy based on the consent of, and the perceptions of, other members of the organization.*
>
> **Principle 3**: *Influence depends initially on trust—on the belief by all members that others are seeking mutual benefit rather than maximizing personal gain. Without this basic trust, persuasion is impossible, because everyone assumes that others are trying to "put one over" on them. A system stressing influence must have a higher level of internal trust than one based on command and power. TH major source of this kind of trust is interdependence: and understanding that the fortunes of all depend on combining the performances of all. Specifically, in a business, it derives from an understanding of the ways in which different pats of the organization contribute to the accomplishment of the overall strategy.*
>
> **Principle 9**: *In order for a system of influence to function, there must be ways of verifying and publicizing reputations. There must, therefore, be unusually thorough and open processes of association and peer evaluation, so that people get a relatively detailed view of each others' strengths and weaknesses. Perhaps the best example of such systems in industry is in investment banking: in many such firms people work in a wide variety of peer teams and are constantly involved in mutual evaluation.*

Introduction

Within emerging occupations such as Radiology and Programming where professional roles within organizations have not been resolved between competing occupations looking for control over the emerging field (Barley, 1990) or when new

technologies are introduced into occupations (Zetka, 2001), past organizational ethnographies have found that in the absence of formal role definitions and norms of conflict resolution, workers negotiate their own roles within organizations (Morrill, 1991). In this chapter the norms of role negotiation and conflict resolution that were developed by members of Star Linux and LithiumBSD will be considered in detail. This chapter will compare the different processes of role negotiation and conflict resolution in each group to determine common practices and possible reasons for differences between the groups.

It is important to note that although members of both groups attend technical conferences that allow them to meet face to face, most communications, and therefore the process of 'role negotiation' in both groups, occurs through electronic communications such as email listservs and IRC. Despite the lack of face-to-face interaction in these groups, past research has indicated that role negotiation and the communication of status are possible within a virtual context (Galeger, et. al., 1998; Easterbrook, 1994; See also Madanmohan and Navelkar, 2002 for one ethnographic study of roles in open source groups).

The apparent conflicts between the quotes above regarding leadership and the 'openness' or 'elitism' of the groups will be explored in this chapter. One major effect of decentralizing power within LithiumBSD and Star Linux is that every member of the group must become a manager of both his contributions and the contributions of others to effectively contribute to the project. Since roles and tasks are not delegated in either group, members must negotiate their level of influence within the project, and convince other members that their technical approach to problems makes sense. This chapter

describes the process of gaining membership and status within the two groups, as well as the methods of negotiating conflicts that have evolved in the absence of formal processes of conflict resolution.

As the opening quotes have also indicated, there appears to be a significant difference between LithiumBSD and Star Linux in their process of determining which individuals can fulfill important roles within the group, although both groups are generally considered to be 'meritocracies' by their members. This difference will be briefly outlined here and discussed in more detail in the 'Role Negotiation' section.

Members of LithiumBSD were more likely to note that it was difficult to gain 'ownership' within an important area of the LithiumBSD system[54]. For example, one member of Star Linux provides a typical description of the method for members becoming a new 'package owner':

> There's nobody who has the job of deciding; there's a first-come-first-served approach, which works well enough most of the time. If several people want it then they get to argue among themselves - co-ownership is one possible solution. I'm not sure there's ever been an irreconcilable problem over this.

Similarly, another member of Star Linux explained that 'package owners' typically perform relatively simple tasks, when their work is compared to the crafting of new programs and as a result the barrier for entry is lower than for LithiumBSD:

> I think of maintainer as someone who doesn't do much coding, which is what most Star Linux people are... they just package stuff up, and make it available to the world. Some people do heavy lifting, or write code for Star Linux like the installation system and the boot disks.

In contrast, all members of LithiumBSD can modify the kernel of the source code itself in

[54] See Chapter VI for more detail on 'ownership'

addition to external packages and as a result there is a higher bar of trust and technical standards required to join this group and as a result, a higher level of status. However, status in Star Linux is also dependent on 'complex skills,' even if the skills required to be a member of Star Linux are less 'complex' than the skill level required to join LithiumBSD. As one member of the group explained:

> There's definitely a continuum between the packages on which significant development work is done within Star Linux itself through to the software that just get rolled up into a Star Linux package and nothing more needs to be done. I maintain things at both ends of this spectrum. (socially, it's considered good for maintainers to be more familiar with their packages, though.)

One support for the hypothesis that a member's status in the group depends on her level of complex technical skills is the fact that both groups have multiple levels of membership with differing standards of merit for gaining 'commit access' to different parts of the system. It is possible to join both Star Linux and LithiumBSD relatively easily as a 'documents committer' who is responsible for editing and translating project documentation, but generally this level of membership garners the lowest level of status in the group. The next most difficult level of membership to gain is the position of 'ports' committers. These individuals, who modify or 'port' external programs such as internet browsers so that they work in the operating system, have a greater potential to 'break the system' (e.g. by adding code to an important program that causes the operating system to malfunction) and this level of status is more difficult to obtain. Finally, the most highly regarded and difficult to join teams are the 'source committers' who perform commits to the kernel itself. Since this group has the ability to modify the core of the operating system, they have the greatest potential for introducing errors that effect the operating system as a whole and this team is the

most difficult to join. This is supported by the following statement by a member of LithiumBSD:

> If you become a ports committer and you're responsible for your app. All you're responsible for is knowing that your application is working in our framework . . . and if it crashes the system, I'm just not going to install the port into my system. . . But if you are a source committer and you break the build, you're going to get the pointy hat and get yelled at by these very vocal developers.
>
> But we're not going to let you become a source committer unless you prove that you have what it takes, and what it takes means that you prove that you have either submitted a number of patches and you've shown that you've submitted some quality and you've prove that you have what it takes. So you'll have a commit bit and a mentor and that mentor will be me and I'll review all of your commits . . . after awhile if I didn't find any problems and people see that you're doing okay and that I trust you, you'll be on your own.

The higher level of role conflict within LithiumBSD, which was noted by interviewees, appears to be due to the higher level of complexity that typical members of the group are engaged in. Members in LithiumBSD are able to modify the core of the operating system itself—the *kernel*--which is essentially a large software package which regulates the processes of the operating system while the Star Linux distribution only adds programs and user interfaces (e.g. to ease installation of the operating system) to the kernel developed by the Linux team, headed by Linus Torvalds. As a result, members of LithiumBSD are essentially competing for 'ownership' of parts of a single software package while members of Star Linux have thousands of packages they can chose to integrate into the Linux kernel. In addition, members of LithiumBSD are performing more complex tasks that can affect the operating system as a whole while Star Linux 'packages' can usually be removed from the operating system until all important

bugs are fixed. This difference in ‘task complexity’ between LithiumBSD and Star Linux appears to explain, at least in part, a tendency by members and other observers to see Star Linux as a more ‘open’ organization and LithiumBSD as more ‘elitist.’

One piece of evidence for this hypothesis is the relatively open nature of the LithiumBSD ‘package team,’ when compared to the LithiumBSD ‘kernel team’ which like Star Linux, incorporates thousands of external programs into the operating system. Similarly, complex decisions in both groups are typically made by expert members, often ‘behind closed doors’ rather than the groups as a whole. A member of LithiumBSD explains this type of decisionmaking and the rationale behind it:

> If you want to get involved it requires a significant amount of time just to know the software and to contribute in a meaningful way. It's not something that you can just decide one day that you want to do that and you can see an impact on what you've done. In terms of technical discussion I think there have been a few on the mailing list but I assume that most of that happened behind closed doors and the gurus involved hammer out the issue.

B. Role Negotiation and Conflict Resolution

1) Initiation into Groups:

The process of joining both Star Linux and LithiumBSD is similar: prospective developers must submit source code for the project to current members of the group for peer review and seek out a ‘mentor’ who agrees to help them learn group practices. If the programming skills of prospective members are deemed up to par, and they demonstrate the ability to make changes suggested by their mentor, potential members must next have

their identify verified in person by a current member. Finally, new members must provide evidence that they understand the policies and philosophy of the group. In Star Linux, there is a formal 'entrance test' to demonstrate that new members understand group practices. A member of Star Linux explained the process of joining the group and gaining status in the following way:

> Applicants apply for developer status through the website, and are then put through a series of tests (technical as well as ideological), and finally, if deemed competent, granted developer access.
> How does the group determine that someone is ideologically committed enough to join? When a new applicant applies to become a maintainer, he is assigned a New Member Manager who takes care of determining exactly that, as well as a lot of other things (among these, confirming the identify of the individual applying).

In LithiumBSD, testing the prospective members for internalization of group practices and culture is left up to mentors to enforce as they see fit. If these mentors decide to add a new member, they inform the leaders of the decision and the leaders decide whether or not to accept new members. No cases were reported of leaders overturning mentors' decisions.

In both groups, completing the initiation process typically takes several months and in some cases more than a year. In Star Linux the delay is explained in part by a slow acceptance process which is held back by the group members who administer the application process and in LithiumBSD the delay is generally caused by a requirement that new members demonstrate a high level of technical skill. Prospective members of LithiumBSD must typically submit dozens of patches which solve current bugs in the operating system to current members who may or may not use or even acknowledge them.

Applicants typically must also verify their identity through a current user, a requirement that serves two purposes: ensuring that new members are who they say they are and meeting another member of the group in person (generally their mentor or 'sponsor') which allows current members to better determine whether the applicant can be trusted to join the group. As a member of Star Linux explained:

> The "web of trust" is one attempt to ensure that developers can be trusted: new applicants are required to have sponsors who vouch for their identity and skills, but this can be difficult in geographic areas where no one develops Star Linux.

Aspiring members of both groups need to make an effort to interest a current member in mentoring them. This typically involves writing technically impressive code and making an effort to get their requests for a mentor noticed among the many messages posted on group mailing lists. This process may not always provide a fair or standardized process for joining the group. For example, one applicant to LithiumBSD reported that he had submitted patches to two different developers who did not acknowledge his changes, and made the changes to the source code that he had suggested, but in a slightly different way.

However, this high barrier of entry in both groups insures that new members are highly committed to the practices and 'ethic' of the group they want to join. In Star Linux, prospective members are asked to demonstrate technical knowledge to current members, as well as the Star Linux 'philosophy' and organizational. New members demonstrate their knowledge of these organizational principles by restating these principles in their own words as part of a detailed test on organizational practices. Prospective members are then required to submit at least one finished 'package' to their 'mentor' which must pass technical muster, and new members are often required to make

changes suggested by their mentor. Through this process, new members demonstrate both required technical skills and an ability to fix problems based on the suggestions of other members. In LithiumBSD, the process of submitting new patches ensures that new members are able to not only demonstrate the necessary technical skills, but that they can adequately communicate their changes on the mailing list to other members and defend them without 'flaming' other members of the group if their changes are challenged. As a past leader of LithiumBSD explained:

> Hopefully, by the time they get to [the point of deciding whether applicants can join the group] we know them. Sort of through interacting with them on mailing lists, you get an idea of what you're like.

Both groups allowed for an indefinite period in deciding whether an applicant should be allowed into the group. LithiumBSD applicants are typically expected to submit 5 to 10 patches[55] before they are accepted as a member of the group and patches can vary from a quick fix of a few lines of code to days of work for the potential member. The important standard in accepting these patches is that a current member, who is an expert in the area that the patch applies to, believes that the changes improve the operating system. Similarly, applicants to the Star Linux group often face an extended waiting period, despite the standardization of the New Member process. In the past, members have waited up to eighteen months before approval decisions were made regarding their applications for membership. Both of these gatekeeping methods ensure that only the most committed, if not the most technically skilled, members join each group. A senior member of Star Linux expressed his mixed feelings with the New Member process as follows:

[55] This number can vary widely, depending on whether the mentor believes the applicant is up to technical par.

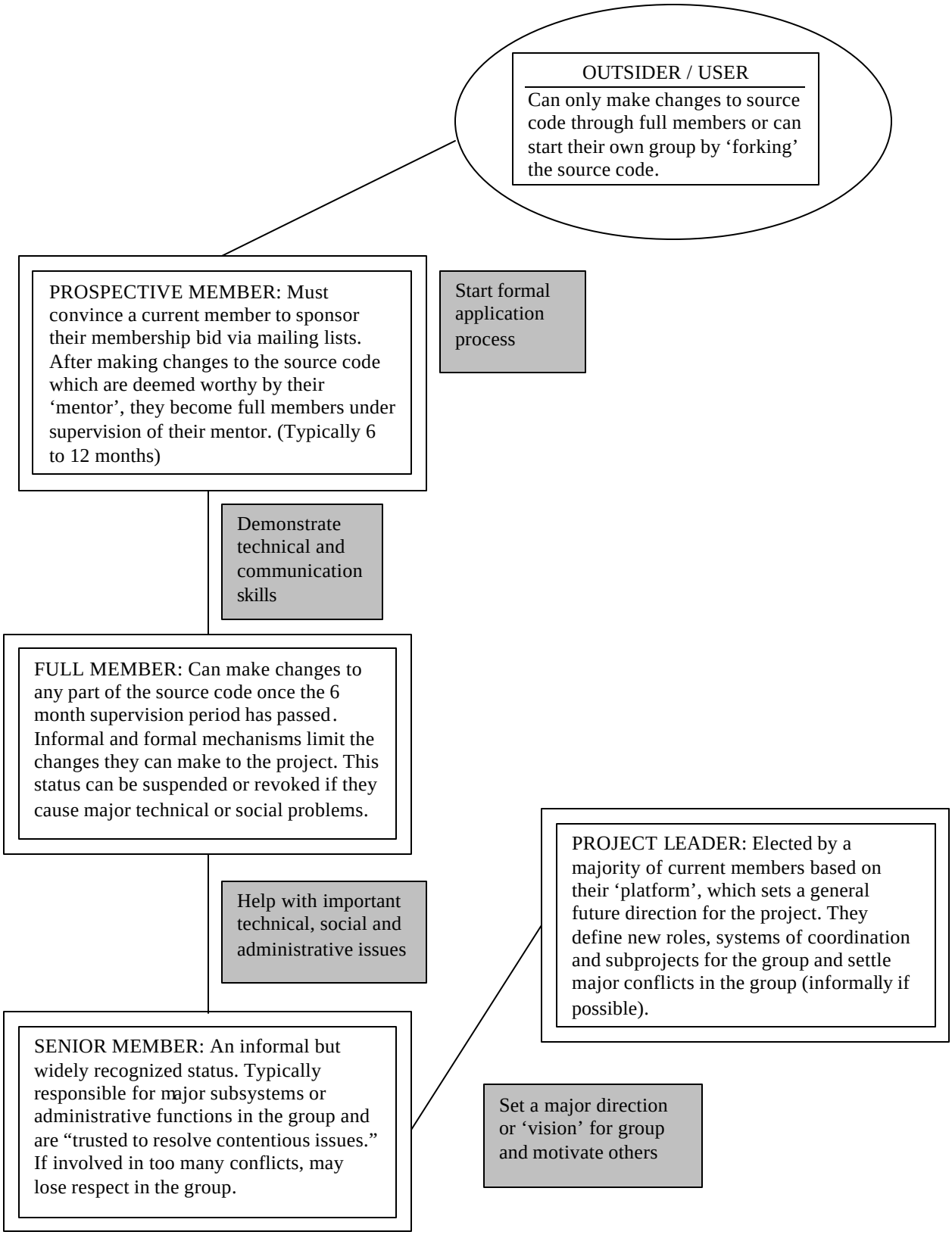
> This certainly means that the barrier is higher because the people who are getting in are more dedicated, because I wouldn't have spent a year and a half just to get in just to get into Star Linux. But on the other hand people say that they still aren't ensuring that the people who are joining Star Linux aren't as skilled as the original group.

Developing an effective process of certifying and socializing new members is an essential function within 'clans' (Ouchi, 1980). Ensuring that new members possess the technical and social skills necessary to effectively contribute to the group effort is necessary for the survival of post-bureaucratic organizations, which rely on the initiative of employees to find and solve organizational problems while effectively coordinating tasks with other group members.

2) Role negotiation

It is the nature of open source code that any interested individual can make changes to the source code for a program that she may want, so it might be expected that there would be conflict over roles within open source groups because it is very easy to take the source code and 'go it alone.' However, to actually have those changes incorporated into the official 'distribution,' the current 'gatekeepers' of the group must accept those changes, unless the individual become a full member of the group with the power to make the changes herself. The diagram below explained the various roles in both groups as well as the successive steps required to gain membership as well as a leadership role for both groups. Generally, gaining status in both projects requires that members demonstrate the ability and willingness to solve important technical problems as well as demonstrating project management skills such as coordinating their work with other developers and resolving technical and personal conflicts.

DIAGRAM 2: STATUS LEVELS WITHIN STAR LINUX AND LITHUMBSD



The membership types differ within LithiumBSD and Star Linux in ways that reflect their organizational philosophy as well as the nature of work within each group. The two groups have developed very different systems of organization, although in both groups there are flexible mechanisms for organizing responsibilities between developers, which is essential in projects that rely almost entirely on the contributions of volunteers.

LithiumBSD requires that members only make changes to one of three areas, depending on their skills and interests: either to the kernel of the system (which controls the majority of low-level processes such as allocation of system memory), to external programs which were modified or 'ported' to work with LithiumBSD, or to project 'documents' such as user manuals or web pages. Members of the group can make changes to any part of the program that they wish, but they are generally chastised if they make changes or 'commits' in an area outside of their expertise, especially if they are only accepted as members trusted less responsibility, such as web and documentation members[56].

In addition, as described in the discussion of coordination systems in the previous chapter, LithiumBSD has also evolved a list of senior developers who are responsible for or 'own' different areas of the system. Members informally negotiate these roles and if a current owner of an area of the program is not able to contribute to the project any longer, there are typically several other members who can take over his responsibilities. Most members view the negotiation of ownership of different parts of the system as based on merit, however it can also lead to conflicts between developers. In at least one case, a

[56] Generally, it is encouraged for 'kernel members' to write documentation for areas of the operating system that they are responsible for as long as they have some technical writing skills. LithiumBSD recognizes that writing documentation and web pages also requires specific skills that good programmers may not posses.

senior member nosily left the project when he was not awarded a leading role within an area of the project where funding was available to pay for the developer selected to lead the project.

New members of LithiumBSD are typically required to clear all their changes with a more senior developer, especially if they are working on important systems in the group. One member explained the process for gaining autonomy and status within the group and how it can be difficult to get changes accepted by the group in areas where there are no experts to review them:

> Basically, you build a track record. Developers like to say "ownership is demonstrated, not demanded." If you're working on some area where there is already someone who has done a lot with it, then you have to write up some patches and get them to sign off on them. And then you have to follow through, and do all the work. And people have to see you're doing commits and not *breaking* anything [causing another part of the operating system not to work properly] while you're adding whatever you're adding.
>
> If it's something where there is no "current owner", then it can be a bit trickier. You may find lots of developers who are uneasy that you're changing some component, but you really don't have anyone who wants to sit down and watch what you're doing. So, you'll get feedback like "I'm uneasy about your change, please don't do it". I had that problem with a simple (two-line) bug-fix that I wrote up for the 'make' command. I had a bug fix, which worked, but there were a few developers who simply were not comfortable with me changing 'make' because it was such a key component to the LithiumBSD build process. So, they were worried that if my change really screwed things up, it might be *very* painful for people to get around whatever bug I introduced. They didn't want to audit my change themselves, they just didn't feel comfortable with me making it. So, there I was, left in limbo.

Both groups also allow any member to make changes to any part of the source code that they wish (although group norms limit this in practice) and this can create problems where one person's changes conflict with those of another developer. As a member of LithiumBSD explained:

> Well obviously all that work isn't going to happen in zero time and you don't know who else is taking a snapshot of the code and working on it at the same time, so basically you have the notion of "patch time conflicts" where nobody objects to the code but you're changing the exact same line I'm changing for a different reason.

As a result of having a minimum of traditional management structures, such as group meetings and well-defined hierarchies of responsibilities, systems of communication can break down. A member of LithiumBSD explained that:

> If this strikes you has kind of hit-and-miss, it is. Sometimes you try to find everyone interested, but you don't happen to catch the eye of someone who is interested. So you finally make the change, only to get a bunch of flak when that person sees it. This can be very frustrating. Other times, you'll rush in to fix something, and someone else will fly off the handle because you've interfered with something they are doing, or they want to do, or they think they want to do when they have the time to do it (which may not be for years).

Since there is little formal structure within LithiumBSD to organize responsibilities within the groups and because every member has the ability to modify code in any area of the system (even if it's frowned upon to work outside of their area of expertise), developers occasionally get into conflicts where they overwrite changes that they disagree with, only to find that they are over-written again by developers advocating competing approaches. These 'commit wars' are described by the following group member as an expected part of a project where "you have hundreds of people changing files all over, millions of files." One member described what happens when system of coordination work and negotiating conflicts break down:

> One person makes a change, the next person makes a commit which overrides that change and the first person commits again. So you're in a war over that section of the code. That unquestionably happens. The feeling of the project is that this is really, really bad because the two people are getting upset. Every time the other guy makes a commit I'm more upset. I'm just getting burned and he's getting burned so he makes

> another commit. Immediately people start choosing sides as to whether Joe is better than Rich and so it sort of draws people into the fight and the real problem with that is if there is an active commit war going on then the people who are drawn into the fight are usually drawn in because they like Joe better than they like Rich and not because Joe has a better implementation than Rich does. So when a commit war flares up you have a lot more polarization based on personality as opposed to technical merit of the code.

"Commit wars" demonstrate one case where the group norms of implementing only the best written code breaks down, and developers circumvent the process of proving the technical superiority of their code and convincing others on the mailing lists that their approach is the best. These conflicts are highly disparaged by the group. For example a high-status developer who recently lost his ownership bid did so because he repeatedly made changes to the kernel after little discussion with the group as a whole or the individuals responsible for the areas that he made changes within.

The vast majority of conflicts in LithiumBSD are over technical issues, including the best approach or solution to a critical problem facing the group. The preferred method for resolving these conflicts is to have both sides implement a solution and then use technical benchmarks to test which solution is performs better. This is not always possible for complex or long term problems, in part because working solutions for competing approaches cannot be easily created to submit for testing and in part because conflicts are often over competing group priorities such as portability (the ability to work on multiple systems, which has the downside of requiring more programming to allow compatibility) and quick release cycles for new versions of the software[57].

One difficulty that arises from the lack of central control over group roles and responsibilities is that important roles may not be filled. LithiumBSD has responded to

this problem by attempting to delegate important administrative tasks to other members, although some members have resented the lack of openness in this process and indicated that they would prefer if the group as a whole had a greater say in the process of fulfilling important roles. As a member of LithiumBSD explained:

> Administrative roles . . . are often left unfulfilled. The leaders have increasingly delegated more power by creating roles such as the Review Committee to arbitrate technical disputes, a function that they used to carry out themselves. Even so, they appointed the Review Committee without any input from the developer community, a point of some dissension within the developer community. Yet, since the group is largely politically naive, the authority of the Review Committee is largely undisputed. Examples of roles that are left unfulfilled include the crucial PR and marketing functions.

Roles in the Star Linux project are very different, but this group also relies on informal norms of role negotiation. In Star Linux, developers 'own' different programs or 'packages' within the system, which means that they are responsible for fixing any problems that occur within their program, however this system can easily break down if the 'owner' of a program shirks his responsibilities. Generally, owners 'win' technical debates because they are considered to be the technical experts regarding their own package. A senior member of Star Linux who currently heads the Issues Committee, which is charged with formally resolving technical conflicts, explained that only once in his memory did his group override the decision of an 'owner' and this decision was highly controversial within the group. He explained the typical process of resolving technical conflicts in the following way:

> Sometimes the people with the best technical argument win. Discussions are easy if everyone knows the problem, there are few solutions, all of them are know or easy to understand, and easy to solve them. Usually it

[57] Interestingly, the two major 'forks' of the BSD project involved exactly these issues and these competing teams each pursue their differing priorities (at least temporarily) in isolation.

> doesn't matter which one is chosen, the big issue to choose one method, and everyone do it the same way. . . sometimes a single person has enough credibility so he can do something by only putting down a proposal, and people accept it. "Best technical argument" is very difficult to determine: unless you know the x5 package very well, it is not easy to be sure if something is better or worse than some other solution. In cases like that usually the owner wins: there are few people who are willing to use a package, and there is no one who is willing to live without the package. So the maintainer can generally propose things for their package, and implement them without many problems.

Star Linux has evolved norms for developers to temporarily or permanently take over the responsibilities of other developers by making “Changes by Non-Owners” (CNOs), a process that typically creates little conflict between members as long as the ‘transgressing’ member is seen by other members as being motivated by a desire to improve the operating system, rather than a desire to make the ‘owner’ look bad or to take control over part of the source code. These temporary transgressions into areas of ‘ownership’ of other developers are common during ‘bug kill’ sessions. A few developers who have a reputation for handling CNOs diplomatically have also organized a Quality Improvement group, and the group generally accepts their right to make CNOs. As a member of Star Linux explained, these individuals tend to have a high level of status within the group, due to their technical and social skills:

> People are more accepting of [changes to the source code made by] "senior" people (although the word "senior" starts a flame-war if you use it in public). People are more accepting of people who have demonstrated skills. . . People are more accepting if someone has a history of being easy to get along with like Carlos. Carlos scores well on every criterion that will make CNOs be accepted.

Developers can also ‘orphan’ projects if they no longer have the time or interest to be responsible for them and other interested members can choose to take over those programs. If a new project become available or ‘orphaned’ because a developer no longer

has time for it, generally the first developer to request to become its owner is given the responsibility[58]. In some cases co-ownership is instituted if more that one developer wants to maintain the same project, however Star Linux generally uses a "one owner" system. Combined with CNOs this ensures both that there is a responsible person for each important part of the system and a method for circumventing that person if they are not adequately maintaining the program. The only downside to this system, which is inherent in any large volunteer project, is that there are generally a large number of 'orphaned' projects, which have no one responsible for fixing bugs within their code. However, this provides a way for new members to prove their value to the project by successfully fixing bugs in programs.

As described in the last chapter, while formal systems such as the 'ownership' of different parts of the operating systems are essential to coordinating the group, these systems also require methods such as CNOs and 'orphaning' the packages of unresponsive members to circumvent these systems when 'owners' fail to live up to their self-assigned responsibilities. Tension exists within both groups between systems of formal roles and responsibilities (termed 'ownership' in both groups) and more informal systems such as CNOs which allow member to ignore these formal role divisions in order to solve group problems. A member of LithiumBSD explained this tension with the concept of 'ownership' within the group:

> This has been kind of a controversial thing in the past because some people have treated maintainership as a lock on some part of the code like a device driver and will say "anybody who wants to commit has to run it through me first." Well that doesn't really work because if I declare this for some random driver, what gives me the right? We've had instances

[58] Only 2% of all programs are formally orphaned, however anecdotal evidence suggests that the percentage of packages which nominally have a member responsible for them who has been inactive for a long period is much higher.

> where exactly this happens and others challenged it. Many flamewars followed and we kind of settled on this kind of advisory kind of statement where basically you can say that you have interest in a piece of code or some part of the tree and you can request that you get review of something before they commit it, and this is strongly encouraged but it's not required. . . If you try to make a change to my device driver and you send me a patch and you ignore me for two weeks, that shouldn't prevent you from eventually getting the code in if I'm totally non-responsive. . .Somebody phrased this as "ownership needs to be demonstrated, not claimed."

Similarly, members of Star Linux explain that even 'senior member' status does not protect members from these informal systems for circumventing formal roles. A member of Star Linux explained that:

> There's definitely some amount of respect for ore senior developers when it comes to CNOing, but there's a tendency to eliminate this. Some of us "newer" developers who have seen a lot of people just go AWOL without saying a word have grown tired of the waiting game, and these days we don't encourage endless patience as before. Over the last few years we've been organizing this "Quality Improvement" group, which never got constituted really :) but the term stuck in people's heads as "the people who can handle a CNO without pissing anyone off." These days we have a technical extension to the upload system. The 'Delayed' directories, where packages can get uploaded first, and then accepted <number> of days later and during this time, if the owner awakes, they can remove the CNO and build their own, so everything's fine. In general, nobody likes the people who are excessively territorial towards their packages.

3) Negotiating High Status Positions

Important positions within both groups tend to be created by members who are willing to do the work, rather than delegated by leaders. The converse of this lack of managerial control also holds true: group leaders not only do not delegate positions, but they rarely expel current members of the group. As a member of LithiumBSD responded when asked whether new leaders ever replace 'hats' [influential positions in the group which form an important part of group infrastructure]: "There hadn't really been much of

an opportunity to do that. We've only really had three groups of leaders . . . and when we came in we very much adopted the attitude of 'we're not going to try to fix anything that isn't broken.'" A current leader of the group agreed that there would be a high level of negative feedback from other members if he ever tried to 'fire' a member holding an influential position in the group.

High status members of Star Linux and LithiumBSD typically have few formal powers not available to typical members. Project leaders and 'senior developers' are generally notable not for their ability to directly change source code (an ability available to all members), or their ability to make major decisions which effect the group as a whole, and in fact they have little formal decisionmaking power in the group. Rather, group leaders can generally only propose new directions for the project as a whole, and must rely on their informal power in the group to implement their plans. A member of Star Linux explained that leaders can only "cajole, request, or complain."

High status members are members who are best able to motivate other members to get involved in new subprojects, the fixing of bugs or other changes and work to solve these problems themselves. As a member of LithiumBSD explained:

> Fixing bugs (AKA "Closing Problem reports") makes you look like a good guy. No question about it. But it's usually a lot of work for less of a payback. Most developers would rather take some project that they can really sink their teeth into, and make some change that is big enough to brag about (so to speak). It's much less fun to spend a few hours to track down an obscure bug in some little thing, particularly if it isn't you who is hitting the bug.
>
> While it makes you look like a good guy, that only happens if you're fixing it "The Right Way". You'll lose some respect if you're constantly applying little band-aid fixes when the general consensus is that the "real" problem is something much larger. Every once-in-awhile some developer will get frustrated at how many PR's are just sitting there, and will stir up a campaign to get other developers to tackle them. This usually has a very

> good result for about a month, and then everyone goes back to working on whatever "new stuff" they want to work on.

Similarly, a member of Star Linux explained the process of gaining status:

> Status in Star Linux is linked to level of competence, and assumed responsibility. If you assume responsibility of some major subproject (and handle it well), your status will obviously rise. Being a lowly package maintainer[59] (and doing nothing else) pretty much leaves you with the masses. A major subproject would be "Porting Star Linux to a new hardware architecture", or "adapting Star Linux to work with the desktop."

This lack of formal power for leaders is especially evident in Star Linux, where members emphasized that any member who has a useful idea and is willing to do the work to organize others can quickly become highly influential in the project. A long-time member of LithiumBSD emphasizes that members of the group only gain status for their current work within the group, and not past accomplishments or skills and that, as a group, they can be quite influential in LithiumBSD:

> In my case I came to LithiumBSD, I had already been a programmer for about 20 years so I might come in and think “well I’ve got 20 years of experience who’s this clown” and the fact of the matter is the LithiumBSD project as a whole recognizes that Chris is a very valuable person who makes a huge contribution just by annoying people like me, a new comer. So even if I’ve got 20 years of experience, I’m not going to win an argument with Chris. . . Now if I become a leader that’ll give me a little bit higher status than people who are not leaders, but not much…. Certainly the claim is that individual leaders are not necessarily more significant than any other developer. But certainly the leaders as a whole votes on something, than that’s much more significant than everybody else. And if you don’t like it, if everybody else doesn’t like it, then they kick the leaders out.
>
> So you have these few people who have a long track record with LithiumBSD and will always have more track record, more prestige and you have the leaders which as a group are very powerful and have a big say in what is going on. And then the way it works out, it’s basically the people who do the most work who get the most credit and get the most leeway to comment on what to the people are doing.

[59] The lowest status level among member of Star Linux. These individuals are responsible for ‘maintaining’ an external program or ‘package’ which is ‘ported’ to the operating system.

Despite formal constitutions in LithiumBSD and Star Linux, there is a layer of informal status including important roles that aren't elected and which don't fall under the control of elected leaders, which are also know as 'hats' in both groups. Many of these 'hats' have been 'grandfathered' into the project from earlier in the project when there were few formal roles. Generally, individuals holding these non-elected roles can only exert a kind of negative power, either in withholding decisions, or by refusing to carry out the roles that they are responsible for[60]. Members of both groups cited examples where influential members of the group delayed in carrying out their expected duties.

These delays appear to serve a 'gatekeeping' function in the group. Two key examples of these 'gatekeeping' roles are individuals charged with formally approving new members and with assigning 'cryptographic keys' to members (which are required to log in to the project to modify source code). In both groups, members have noted long delays in these processes. At one period in time, in the Star Linux project, no new keys had been assigned for over a year and recently the process of approving new members had taken a year or more, but in both cases no formal action was made to discipline the 'hats' responsible or to assign new members in their place. It is likely that these delays were tolerated in part because they almost entirely affected new members of the group.

With a large audience of willing members, this gatekeeping is seen by members as a 'weeding out' of uncommitted members, which helps keep group size more manageable. A long-time member of Star Linux who holds several important roles in the group described the evolution of these gatekeeping functions within the group:

> When the project started, members were more likely to be coding professionals. Now, especially after the New Member process was instituted [which allows anyone to apply for membership], there is a

[60] This is also generally true for formal leaders as well.

> perception that this group of Star Linux members and the backgrounds that they come from has widened considerably from the days of yore. After NM policy was developed, the two people who were handling the New Member process grew very dissatisfied with either the applicants that were going in or the work that they had to do, it wasn't clear because they didn't say what was bothering them, they just stopped letting new people into Star Linux. For 18 months or so there was nobody getting in.

Despite continued delays in accepting new members over a period of several years, little formal action has been taken to change this system. One member explained why generally there is such resistance to taking formal action against high-status members:

> There were some situations when developers proposed [that formal action should be taken to punish 'hats'] but the Group Leaders usually had to talk to all the interested parties to get the real perspective. Arguably the leaders should have replaced some people in some situations even after review, but it hasn't happened for several reasons. First of all, disowning senior officers who are still there means disallowing them from doing the useful work that they have been doing so far, not just the stuff they would be removed for doing. Secondly, it creates a rift, a real conflict which impacts that developer, the team and the developer community overall. ("They removed him, will they remove me too?" would be logical thinking . .) Thirdly, the leader doesn't have the power to do that, because we've never tested how it would work :)

Despite a proliferation of non-elected positions of power in LithiumBSD and Star Linux, the majority of members of both groups believe that their groups function as a meritocracy. While some current member would like these positions to fall under the formal control of project leaders, or for these positions to be elected by the group as a whole, the majority of members that were interviewed did not see a major problem with this system, except in cases of obvious and persistent ignorance of duties or abuse of power. As one member of Star Linux explained, informal positions of influence within the group, including that of 'senior developer' are assigned fairly:

> This division between junior and senior members is basically an unwritten

> distinction. The junior members are those that have been accepted recently and which still require guidance from members of the team that have been on the team longer. Less often, they require guidance in technical issues, and more often in policy issues. It is unlikely that you will ever see, for example, a recently-appointed listmaster go tackle controversial requests for new [email] lists, whereas someone who's been on the group for a while will be expected to have a better grasp of things and be trusted to resolve contentious issues. It's basically a system based on benevolent dictatorship, like the project itself.

While there is no formal definition for whether a member is a 'senior member' of either group, there is usually high consensus regarding this distinction. A member of Star Linux explained that:

> "Senior member" is not a formal title it's impossible to not get it if you deserve it (or the other way around). Also there will be some disagreement as to whether someone qualifies. You treat people based on your opinion of them.

Although there is some competition for high status roles in LithiumBSD and Star Linux, often these roles are assigned to the first person to take the initiative in doing the work that has the technical skills to carry it out. As one member of Star Linux explained: "Often it's not that somebody asks somebody to do something, but that they took it upon themselves to fix something that had been bothering them." Generally, due to the large amount of work to be done in both projects, any member who is interested in tackling a problem is welcome to do so. While officially the process of becoming a owner of a popular program for Star Linux and LithiumBSD is "first come, first served," in practice developers need to demonstrate that are technically skilled enough to handle the task. As one member of Star Linux explained:

> If the current people who maintain base programs [which are essential to the operating system] all had plenty of spare time then almost no one would get a base program. Anyone who's in charge of a base program will probably prefer to give it to someone else who's done similar work before. But such people have hundreds of things that they want to do and are

capable of doing.

However, gaining status within these groups requires more than simply demonstrating the technical skills necessary to complete important work. In the following statement, a member of LithiumBSD explained that even the process of initially joining the project requires an understanding and demonstration of group norms and that technical skills are not sufficient for gaining status in the group:

> How does one go from being "some unknown guy" to being someone who would be recommended as a full member. The most popular way is to send enough bug reports of high-enough quality that other members start to notice that you have the ability to really contribute to LithiumBSD. You must show not just that you would like to be a member, but that you can write code on your own which would improve LithiumBSD. They also notice how you behave on the mailing lists, and decide "is this a guy that we would like to work with?"

Similarly, a past leader of LithiumBSD described the moment when he realized that his past efforts implementing major technical projects within the group were not enough to convince other developers that he deserved to be a group leader:

> I realized afterwards that I had taken for granted that I had assumed too much about the composition of the people of the project. I had sort of assumed that everyone in the project knew who I was and in reality, we were in a period of huge growth in the project. For the months preceding election for the first time, I had actually been totally engrossed in finishing a satellite project that I had been working on and over to Africa to set up a satellite and I had been if not letting things slide at Star Linux, I had not been doing anything publicly visible for a number of months. There was a mistake in the election processing that year and the ballots got published, which there not supposed to. I couldn't help myself, I went and stared at the ballots for a little bit and one of the things that I came to realize was that every person who's name or email address I recognized picked me as either their first or second choice among the candidates and conversely almost everyone that didn’t pick me, I had never heard of. By looking at the growth of the project over the previous year, I realized that the problem was that I wasn't visible enough to these new developers.

To gain status within these two projects, members must demonstrate their ability to resolve technical and personal conflicts through public dialogue on the mailing lists, rather than engaging in less acceptable practices such as “commit wars.” Another frowned upon practice in both groups, “bug terrorism,” occurs when two or more developers fail to resolve their technical conflicts through discussion. This practice is described by a member of Star Linux as involving “the arbitrary filing of bugs as harassment.” Members who engage in these practices, or who ‘flame’ the mailing lists, generally do not gain high status roles.

Members of both groups emphasized that individual status could only come though consistently demonstrating a high level of technical skills and by implementing interesting or difficult projects, or by setting a technical ‘vision’ for the group. Often, members emphasized that leaders took the initiative to define their leadership roles, by starting important projects or making other major changes to the project. As one member of LithiumBSD explained:

> Part of being a group leader is setting direction. I see that as a precursor to a leadership role. Basically you have to take leadership, you can't have it given to you. You have to do something to demonstrate my leadership. Choose a direction, set a project, coordinate the project, and finish it. In that very act you’ve been given leadership and you start another project and you get to the point where the project is larger in scope and you have more direction.

A member of Star Linux provided a similar assessment of the current Group Leader:

> The best example [of leaders solving important problems] that I remember is the current New Member process. It was stalled for a year or two. There was a whole lot of complaining about it. Years worth of complaining. Every so often someone would be told to go do something about it. A few of them actually did. I don't remember the details very well, but I believe Nathan was involved at this stage, in putting together the new New Member process; now he shows up on the Organizational Structure page under that item :) (He also shows up in a few other places, primarily the

> Quality Improvement team and our newly elected Group Leader for this year)

In order to gain status in the group, members must demonstrate their ability to successfully negotiate technical and personal conflicts within their group. High status members of the group are expected to remain as objective as possible in the face of criticism of their ideas, and to not respond overly emotionally if they are personally criticized or 'flamed.' For example, in both of the major conflicts in recent LithiumBSD group history, members who played a key technical role in the group but could not successfully negotiate conflict were formally and informally punished and lost status in the group. In the most recent case, the developer lost his membership in the group after a long history of conflicts with other members on group email lists and in private emails. As a result he now has to submit his changes to the kernel to other developers. The other developer involved in a major conflict resigned the group after a bitter flamewar started when another developer was appointed to a paid position within the group in an area where he considered himself the de facto expert.

4) Leadership

> I'm sure you've heard the old joke: "Managing programmers is like herding cats." The problem with virtual teams is they are the worst example of the "herding cats" problem of which I'm aware. Distributed engineering only works if everyone is highly motivated, self-directed and disciplined. Since no group is composed entirely of people that are highly motivated, self- directed and disciplined, or, at least not all of the time, virtual teams are pretty much a disaster waiting to happen. Now, I realize that some of you have already assumed your sleeping-in-class faces and tuned me out with the simple response, "So what? That's *their* problem, I'm not a manager, I'm hacking on this stuff for fun." I'd like to correct you on that point before we go any further. You *are* a manager. You manage your time, you manage your teammates, and you manage your group leaders. You joined this group because you cared about its goals. In fact,

> since you aren't getting paid, I know that you care more about the groups' goals than most people care about their "job". And, frankly, as a group member, my point-of-view is that I don't want my teammates getting off track, because it lessens my chances of reaching my goals, and maximizes my chance of having wasted my time. You need to understand the engineering management issues in your group if for no other reason than to defend your own self-interest in getting your goals accomplished.
>
> --Keith Bostek “BSD guru”, speaking to members of several BSD groups

> A lot of leadership is just a passionate desire to take the things that I've observed and am excited about and to get other people to get excited about them as well. It's funny, I don't know if this comes across but I'm not one of those 'I want to teach people kind of people kind of people. In fact, I detest the whole process of having to manage people. I'm a leader, not a manager. I guess is what I'm trying to say and it's a lead by example process rather than trying to get other people to do things. In other words it's an 'I think this is cool, why don't you come and work on it with me’ instead of assigning work to people and making sure it gets done.
>
> --Past leader of LithiumBSD

Traditional leadership practices such as delegation of tasks and centralized system of control are not esteemed by most developers. Several members of both groups pointed out that a strong leader who attempted too much managerial influence would be resisted by the group. In some cases, past leaders of the group who attempted too much control over the group burned out due to the large amount of work involved and a high number of conflicts with other developers. A developer in the Star Linux project explained the case of one founding leader who wound up resigning the project:

> He tended to rule by fiat, he made decisions, he took strong stances and in some way he moved the project and set it in the direction it’s going now and he also found out that Star Linux was kind of like herding cats, you found out where the cats wanted to go and then herd them. And he was not interested in where we wanted to go.

Current and past leaders in both groups recounted stories where their attempts to formally delegate roles and responsibilities aroused resentments among members. As one past

leader of LithiumBSD explained, his efforts to better define which positions within the group were ‘delegates’ which by definition fall under the formal control of leaders met with mixed results:

> When I became Group Leader I wanted to go through all of the delegations and ask them how I could help them and say “these actually are the delegations" but when I did that I found out that it's not actually clear if one particular person has been delegated or if he is just doing that and some people asked me "please don't make me a delegate because the next PL might remove me . . .That's something that's not clear at all and that's something that I guess need verification and the last Group Leader didn't do anything about that either. On the other hand, I think that it's working at the moment and I’m not sure what to do about it. One thing that I’m trying to say is that the system we have at the moment works quite well.
>
> What happens now is that people volunteer to work in an area they just do it. I fear that if I make all of those people delegates that I’m introducing some sort of central control or coordination that may not be very good. At the moment it's very decentralized so I’m hesitant to define those positions as delegates because a future GL might reassign those positions. . . . It’s not that the new GL changes everything and defines his own delegates, like in politics.

Similarly, a newly elected leader of Star Linux explained that one of the first things he heard upon winning office was that “leaders don’t change anything” in the group. This perception, which is shared by many member of both groups, seems to have taken hold because leaders make few formal decisions and generally work in the background to plan project goals and resolve conflicts. This perception is exacerbated by a tactic many leaders and ‘senior members’ take when faced by a major decision: to ‘sit on’ the decision and make no formal decision, often pleading a lack of time, until developers are able solve problems and make important decisions for the group themselves.

Leaders find that leading the group is like “herding cats, you figure out which direction they are going and then you lead them,” as several members of both groups

pointed out. This style of management is evident in the approach that a past leader of Star Linux took towards filling important roles in the groups. He explained that:

> Almost never are you successful by picking somebody out of the blue. It's almost always the case where you try to find somebody who cares a lot of about something and then slap a title on them so they feel good about stepping on other people's toes . . . The project leader couldn't just pick somebody . . . There's almost always an apprenticeship process that has to occur before someone has the knowledge and the skills to hope to be successful in that role. Sometimes this delegation is simply the process of setting up the right environment for people to be learning the right skills and getting themselves educated.

The same leader explained that as a result of using this 'invisible leadership' approach, he has had a major effect in setting the technical direction of the group that unfortunately was not noted by the group because it was "probably not controversial enough." However, he takes some consolation in the fact that the candidates that he ran against for reelection all adopted some version of his 'technical vision' for the group as their platform.

Leaders who did not adopt this style of leadership, especially as the project grew[61], faced major conflicts from other members if they attempted to mandate that other users complete tasks, or overrule the decisions of other members. As a senior member of Star Linux explained when asked what would happen if a leader attempted to do so:

> Oh, there'd probably be a terrible stink. If people objected to the action enough, there's a complex vote process for the general body to overturn his decision. And I'd be amazed if it actually came to that; we tend to vote for reasonable people as a pretty high winning trait.

The hands-off style of leadership adopted by most leaders was explained to me in the following discussion with a member of Star Linux:

[61] See Chapter 5 for a model of post-bureaucratic growth which explains how leadership styles change with size.

> C: What sort of influence does the Group Leader and these core developers have in the group? How do they convince others to work on projects they think are important, etc.
>
> D: By being persuasive. Our current GL in particular seems to be a pretty good motivator. In general, they both make suggestions - which are listened to, because of the general respect for the suggestor - and do a large portion of the work. There's definitely a 10% in Star Linux who do 95% of the work!
>
> C: By persuasive, do you mean mainly logical argument?
>
> D: I wouldn't say that's how people get motivated; that's how their direction is changed and focused, maybe. It's mostly a matter of making suggestions, finding interested people and bringing them together with interesting suggestions.

Lacking the ability to 'fire' the mostly volunteer members in the group as well as financial and project resources, leaders must turn to motivating developer to follow their suggestions in more subtle ways. One leader explains that his main method for coordinating the group is convincing them that completing the work would be the best thing for the project as a whole:

> That's not typical for the project leader because the PL doesn't typically have those kinds of developers or resources to throw at a problem. But you can do the same thing by saying 'look, this problem has to be solved this week or we'll have a real problem releasing next week, trying to really get folks to solve the problem.

Leaders typically also avoid punishing other members for not fulfilling their responsibilities or making mistakes. As a current leader of LithiumBSD explained "only when the individual is hostile or pathological should the individual be suppressed because it affects the entire group."

A member of Star Linux explained the group's attitude towards leaders: "Leaders tend to be 'a first among equals' and are often more in the background of the project than

not." Another Star Linux developer who ran for project leadership explained the non-traditional qualities necessary to effectively lead open source projects:

> Accomplished technocrats wield a lot of influence in terms of popularity, but not all such people are capable of constructive confrontation. For example, it could be argued the we've had Group Leaders in the past who were technically accomplished but prone to fits of anger, and those who were technically accomplished but insufficiently assertive to get managerial problems solved. I think a Group Leader needs enough technical aptitude and knowledge to have credibility and respect among the developers. But he or she must also be capable of constructive confrontation, with an emphasis on both words :)

Finally, a past leader of Star Linux explained his personal leadership style in the following way:

> We talked through these things a little bit and he designed a solution and implemented and after that, very few months I'd sort of bump into him somewhere and say 'so how's it going' and get an update from him. I never felt a need to second-guess the approach that he had chosen to take so that's an example where it all works beautifully.
>
> The sense I've had is that there is the opportunity to interact with delegates, there's the opportunity to set expectations, there's the opportunity to get reports on how well they're meeting you're expectations and you always have the opportunity to delegate somebody else. The business of un-delegating somebody is messy. The right way to handle this in almost every case is to figure out how to convince the individual in question that the project would be better off if they did something else.

Positions of leadership in both groups often carry a large amount of extra work with little appreciation from typical members of the group, with one member going so far as stating that they are generally "invisible, and that's a good thing because it means that the system is working well" although he also stated that he would "definitely" put a leadership position on his resume. A current leader of LithiumBSD seemed to agree with this assessment when he explained that "the most visible time you'll see [leaders] is either in times of change or conflict and then we'll be either shepherding change or

resolving conflict" and a senior member of Star Linux explained that “in crass political terms” many senior members have more power than leaders, because they often control crucial infrastructure (such as the servers that house the project source code). If ‘senior members’ ignore their duties, the project will rapidly grind to a halt while the leader’s role in the group is more intangible and includes duties such as motivating other developers to pursue long-term projects, which do not halt a project if neglected.

Leaders also tend to be members who ‘get things done’ that other members do not want to do. A senior member of Star Linux explained that:

> The larger issues of administration and coordination are often neglected; sometimes because there is no one to do them, and/or because no one has a clear mandate to act upon them. This is where I think the Group Leader is required to act; indeed, I consider this to be perhaps *the* central duty of the GL.

Leaders are expected to find members to fulfill important organizational functions, or if that fails, to take responsibility for these tasks themselves if they are essential to the project. Another senior member explained that “I think one of the main tasks of the Project Leader is to coordinate and motivate people. In short, to lead.”

5) Conflict Resolution

The lack of formal roles and the fact that any member potentially has a say in any technical decision within the group, often creates conflicts between members. As a member of LithiumBSD explained:

> Part of being a manager is making compromises to get the job done, and part of what fuels people to work on something like LithiumBSD is they say ‘I’m going to do it right, I’m not going to do what’s fast, I’m not going to do what’s expedient, I’m going to do it right.’ Maybe I’m nuts but this

> is what I think is right. I want to do it and I want to prove that it's the right thing to do. I guess that's a common thread among the LithiumBSD group. There's sort of an aversion to too much management, too much meddling, you don't want to do compromises, you want to do the best technical solution, even if that requires fighting it out. And lately there's been, as I said, a few very good programmers who are just causing too much trouble and right now what's happened in fact just this last month is that two of these leaders have resigned because they're tired of the fighting, they're just tired of it.

The rules that organize the group are both created by the development community and enforced informally by members, and there are few formal procedures for conflict resolution in either group[62]. Only in extreme cases are leaders called to resolve conflicts and when they do leaders generally attempt to encourage discussion between disputants. The most extreme punishment levied by leaders is generally a temporary expulsion from the project for a few days or weeks. Rather, developers are expected to come to a solution themselves after discussing the issue on mailing lists, or through private emails.

One of the few cases where leaders of the LithiumBSD project intervened to settle a conflict involved a high status member of the group who repeatedly became involved in conflicts with other members when he made changes to their part of the system without getting their permission. After the developer community complained over a period of several months, the developer was temporarily suspended from the project and when conflicts continued over a period of two years, his ownership bid in the project was removed, forcing him to channel his changes to source code though other developers. A member of LithiumBSD explained that project leaders became involved because "in this case, as I understood it, it involved the entire developer community and was causing a lot of bad blood inside the project." Another developer explained that this conflict is simply

[62] See the Policy section for more information on rule creation.

a very visible manifestation of a common coordination problem with the group—coordinating peer review:

> This is Will's biggest crime. He would see a problem and fix it, and not take the time to track down anyone who might care about the source code he's touching. And when anyone gave *him* flak about that, he'd give fire right back at them.

By failing to abide by the norms of peer review, 'ownership' and group debate regarding technical changes, this member was eventually expelled from the group, despite his major contributions to the project, although he is still allowed to submit patches through other members. A leader involved in the decision to expel this member explained that the decision was necessary because 'Will' was not able to abide the group norms of peer review and conflict resolution:

> I think this was an unfortunate but necessary solution. I have never had any problems with Will, and he's done some good things for some LithiumBSD users here at OSU, so I consider him a "good guy". I think he's the kind of guy who is a real asset to the LithiumBSD project. That said, he does have a way of "diving into a project", and perhaps running into some section of source code that someone else is actively working on, and that pisses the other person off . . .The thing is, when presented with someone bitching and moaning, he is pretty likely to flare up and make matters worse, instead of maybe finding some pleasant middle ground. So, the people who *want* to cause him trouble can usually get him upset enough that he will alienate some other developers.

High status members, such as 'Will,' who become involved in personal conflicts risk losing status in the group. A current leader of LithiumBSD described a situation where he found himself violating this expectation:

> There was an issue that I was trying to deal with between a core member and one of our 'trouble committers' and I got way too involved and I wasn't paying attention and I got way to personally involved in the thing and I suddenly stepped back and saw what I'd been saying and it had been completely inappropriate. Leaders, we're human. So I told the other (leader) guys "okay, hang on I've just screwed up and see what you can rescue from that."

However, developers are strongly encouraged to debate technical points with other members, which is a necessary component of effective peer review, as long as the debate doesn't 'get personal.' Occasionally, major issues arise in the group that cannot be settled through group debate or 'flamewars' on the mailing lists. A member of Star Linux provided a concise definition of these large-scale conflicts "1) it needs to command the concern of a large number of members, and 2) it needs to be clear that the issue isn't going to burn itself out. It keeps coming up, and it seems impossible to resolve without a formal vote." These large-scale and often project-wide debates are difficult to resolve through peer review and technical debate, often because they balance competing technical priorities (such as portability, or the ability to run on many systems, and release time, or the amount of time required to release a new version of the operating system).

Although formal structures exist in both groups for resolving conflicts, they are rarely used. An Issues Committee exists in Star Linux to resolve technical issues, but it is almost entirely inactive and no formal action has been taken by this board to resolve a conflict between members for over three years, despite a few petitions for them to do so. The current leader of the Issues Committee described this situation:

> The Issues Committee is mostly moribund. The last couple of decisions the committee made was to punt the problem back to the developers, who had resolved it before the committee took action . . . The issues committee should be inactive, ideally, that means there are no conflicts. I think that the inactivity has gone way past that.

This inaction of the Issues Committee illustrates that despite the wide range of technical issues that occur within hundreds of developers working on complex software, few are resolved formally. Overall, there was a surprising lack of conflict in both groups, considering the fact that both groups have very weak systems of formal conflict

resolution. Conflicts that divide the community as a whole are also rare—none were reported in the history of Star Linux and two 'forks' of the LithiumBSD project occurred in the past twenty years, neither of which had a major effect on the group as a whole.

Another member of Star Linux described the informal process of resolving conflicts, which has evolved in place of formal rules:

> If there is a dispute, and there have been disputes about bug severity and whether bugs should be opened or closed, among other issues, the first thing that happens in any Star Linux process is a flamewar and we have flamewars for everything. When we have policy changes, formal resolutions, or pretty much anything, there's a flame war. And if the dispute still continues you throw it to the Issue Committee, who makes a ruling.

The same member explained that generally decisions are made 'by those who are willing to do the work' and that in the long run, developers tend to support the best technical solution for a problem:

> It is important to understand that everyone works for the same goal, there may be differences on the idea of "what's best for the project", but since it is software, and not religion, and most are pragmatic people, there can always be the punchline of: "code your changes and a benchmark-test and prove your solution is better".

The group as a whole also may decide which solution is best after they have tried both options out. A member of Star Linux explained this process of 'group peer review':

> Generally, competing options are implemented and people try them. The one that gets the best regard gets used. Sometimes alternate solutions are proposed but only one of the people is willing to do the coding, which makes for an easy decision. . . . [Members then] try the solution and decide that they want to use it themselves. Eventually one option becomes significantly more popular than the other.

A commitment in both groups to "doing things right" and making decision based on technical benchmarks helps members resolve the majority of their conflicts informally. In the few cases where developers are not able to resolve

issues themselves and a major conflict arises that is affecting the project, leaders will often attempt to encourage the developers to resolve their differences, for example by having them meet in person or by phone. Leaders generally attempt to stay out of conflicts within the group because taking sides or resolving an issue by fiat tends to lead to resentment among developers who believe they are overstepping their powers. A past leader of Star Linux explained this conservative tendency regarding conflict resolution:

> There is sometimes this strong desire to have the Group Leader join a conversation and 'smite the heathens' instead of letting discussions go to their natural conclusions and then let the people who do the work go off and do what they think is the right thing anyway. Frankly, so far I've seen the same kind of behaviors from Nathan, my replacement. Of course there are a lot of months yet where he might choose to jump in and 'smite the heathens' at some point but so far he seems to be taking a similar approach of letting people who make decisions in various areas make decisions and get on with it.

A final form of conflict endemic to a group lacking bureaucratic roles and formal definitions of responsibilities is what the leader cited above called ‘process problems’ where a senior member or leader of the group ‘sat on’ decisions that another member believed that they should resolve. He explained that these conflicts are difficult for leaders to resolve because they lack formal means of censuring senior members:

> Process problems [occur] when somebody is perceived as having authority and someone else has an action that they want to have occur and that they believe requires an action from the person with perceived authority and the person with perceived authority is either ignoring them or telling them to go away . . . The best examples of this have to do with the New Member process where that person has to make the yes or no decision about whether the person becomes a member of the project or not. That person is human so there are sometimes periods of time where no action is taken on a number of application for some number of weeks. People get really frantic when this happens and they really hope that the Group Leader will jump up and down an smite the individual who’s responsible

> for making those decisions, but that is of course a really bad way to motivate someone to make decisions.

It appears that this tendency for leaders not to resolve these process conflicts parallels a tendency by leaders not to get involved in group conflicts: both tendencies appear to be an important part of healthy post-bureaucratic work groups which trust that members and senior members will be better than leaders at deciding conflicts where they are the experts, and at enforcing group norms when other members transgress them.

The diagram below summarizes common forms of conflict in both groups, how they typically are perceived by members of the groups, and the informal and formal methods for resolving them.

DIAGRAM 3: COMMON TYPES OF CONFLICTS AND METHODS OF RESOLUTION

Types of Conflict	Group Perception	Conflict Resolution
Personal Conflicts	Highly discouraged.	Generally, developers are expected to resolve themselves and often one or both members with 'flame out.' In extreme cases, leaders intervene
'Bug Terrorism,' 'Bug Severity Wars,' Commit Wars	Highly Discouraged as they represent the failure of peer review norms	Same as above
Technical Debate	A fundamental part of peer review process	Public and private email between applicable experts. Very rarely decided by formal 'technical bodies'
Group-wide Technical Conflicts	Seen as a necessary part of having an open organization. Often due to competing priorities within group.	Lead to long-term flamewars and may never be resolved. New leaders often provide innovative solutions. May also lead to project 'forks'
Process Conflicts (One member does not fulfill what another views as his responsibility)	Key to leadership tactic of 'sitting on decisions'. Some members resent delays.	Group dissatisfaction with 'inactive' leaders can lead to election of new leaders. Senior member can be informally shamed or otherwise pressured but very rarely removed.

This chapter detailed the norms of role negotiation, leadership and conflict resolution that evolved within Star Linux and LithiumBSD in order to determine individual status and solve conflicts in the absence of a strong management structure that would otherwise be responsible for these infrastructural duties. The findings in this chapter generally agreed with the principles of postbureaucratic organization cited at the head of the chapter. Especially within Star Linux, it was found that individuals rose to influential positions to the degree that they were

able to persuade other members that their viewpoint was worthy[63]. Members of both groups reported cases where highly skilled members who did not master this style of communication, for example by not making an effort to publicize their efforts within the group and letting their work 'speak for itself,' lost influence within the group. Both groups also have well-developed reputational systems which inform members of members areas of expertise and which members can be trusted with important duties. The following chapter considers the effects that the growth of membership within open source groups may have for the software profession.

[63] See the second section of Chapter VIII for a case study of this type of leadership

Chapter VII: The 'Open Source Movement' as an Evolving Professional Community

> The craftsman's work is the mainspring of the only world he knows; he does not flee from work into a separate sphere of leisure . . . he brings to his non-working hours the values and qualities developed and employed in his working time. His idle conversation is shop talk; his friends follow the same line of work as he, and share a kinship of feeling and thought.

--C.W. Mills, *White Collar*, 1956, pg. 223

> Occupational communities are, by and large, those work domains where member identities and work practices have not been fragmented into organizationally-defined positions by highly detailed job descriptions, where work performance is not ultimately judged by a management cadre, and where entrance to and exit form the occupation is not controlled by any one heteronymous organization.

--John Van Maanen and Stephen Barley, 1984, pg. 350

Introduction

Software programming has suffered something of a crisis of professional confidence (Parnas, 1997) although the professional status of programmers has been unclear since the birth of the occupation (Loeske and Songuist, 1979; See also Barley, 1996b for a general discussion of technicians in organizations). Wyatt Gibbs found in 1994 that over 80% of large-scale software projects were considered 'operational failures' that either did not meet project specifications, or failed to work at all (Gibbs, 1994:87). Robert Kraut and Lynn Streeter point out that the typical process of planning and designing software fails to take into account the high level of uncertainty within the software development process, which requires constant communication and coordination between software developers working on highly interdependent 'modules' (portions of the project source code that fulfill a specific function such as displaying output on the

users monitor). In addition, Kraut and Streeter point out that often unexpected design flaws only become evident in the process of programming, making it difficult to plan for and manage the software development process (Kraut and Streeter, 1995). The difficulties of applying 'scientific management' to programmers was evident almost from the dawn of computing (Greenbaum, 1979) and despite efforts to deskill programming (Kraft, 1977) and to restructure high-tech organizations to make programmers more efficient (Schellenberg, 1996), managers of software projects have found 'no silver bullet' for managing programmers (Brooks, 1987; Brooks, 1978).

Members of open source groups appear to be both in the process of developing a new professional approach to programming, and to emerging 'best practices' for software development. In the past this professional knowledge would reside within specific organizations that had a strong interest in not sharing their development methodology with other competitors. As a result, programmers were typically educated only in the technical skills needed to write the programs, and not the skills required to coordinate large scale and complex programs with other developers, which remained locked in the head of experienced programmers. The response of the software industry to this lack of professional knowledge regarding the actual process of crafting software has been to develop detailed methodologies to define client requirements and to allow managers to better track the software development process.

While these methods are useful in clarifying the functionality of the software, they provide only a basic blueprint for design and do little to resolve conflicts that arise during the development process. These models assume that a static software product is being developed, although it is much more common for the product specification to shift,

due to changes in client needs, user feedback, or problems in implementation. Typical methods for planning software design rarely incorporate these external factors and that may explain the high failure rate that most large-scale software design projects. One observer of the typical process for planning software development explains that methods of planning software projects are an imperfect solution for designing software because they do not allow for informal communication and conflicts that arise between developers:

> Typical requirements analysis methods are geared towards the development of a consistent specification [of the end product]. They do not allow conflicts to be expressed, let alone constructively resolved. Indeed, we could characterize existing methods as conflict avoidance, in that they prescribe particular approaches, which assist software practitioners to break problems down and resolve design decisions in particular ways. Uncertainty is reduced by the provision of the collective wisdom embodied in the methodology. . .
>
> Software design necessarily involves organizational change, and even end-user involvement in the design process does not remove conflict. Two obvious sources of conflict in requirement engineering are conflict between the participants' perceptions of the problem, and conflict between the many goals of a design. Other sources of conflict include conflicts between suggested solution components; conflict between stated constraints; conflicts between perceived needs; conflicts in resource usage; and discrepancies between evaluations of priority (Easterbrook, 1994: 42).

The practices that open source groups have evolved for assigning tasks and resolving conflicts between developers, which were explained in the prior chapter, appear to be an alternative model for developing software in contrast to typical 'software requirements' planning. As open source software and development practices gain increased acceptance among companies and other organizations, this alternative model of software professionalism where software developers hold an important role in resolving technical and design conflicts and in deciding the 'technical vision' of organizations, may become increasingly adopted within organizations who are looking for alternative models

for managing 'intellectual labor.' While typically organizations find that at least two distinct cultures exist within organizations: that of managers and workers (Santino, 1978), the open source movement appears to be an evolving 'occupational' division of labor for programming which resists the top-down control system typical of the bureaucratic division of labor[64].

While 'administrative' or bureaucratically organized divisions of labor define roles and tasks top-down, occupational divisions of labor allow workers to decide the roles and methods that are best for accomplishing work tasks. The increased complexities of intellectual labor combined with an increase in work teams and 'outsourcing' of software development has challenged the dominance of managing programmers though an administrative division of labor which makes programming increasingly difficult to supervise and coordinate. James Zetka, Jr. explains that pressures of globalization of labor and the technical complexity of work have led to an increased pressure on organizations to adopt an occupational division of labor (See also Adler, 2001; Winter and Taylor, 1996). He characterizes this organizational logic in the following way:

> Such a [division of labor] decentralizes authority; workers themselves make task decisions. Through intensive socialization workers develop distinctive work-centered identities and value systems and become committed and involved members of moral communities, internalizing their standards and practices. The collective understandings that make up the occupational lore of these communities ultimately guide workers' decisions. . .
>
> More complex occupational divisions of labor develop mechanisms for extensive task coordination . . . Such coordination differs fundamentally from that occurring in bureaucracies. . .it typically involves informal communications among workers doing the job. It is collaborative, not command driven. This nonhierarchical mode of coordination works effectively because of the shared understandings that structure occupational members' values and guide their workplace activities. (Zetka, 2001: 1496)

[64] See Stinchcomb, 1959 for the classic definition of occupational or 'craft' models of work

The occupational division of labor for software development which is evolving within the 'open source movement,' and which is challenging the traditional model of professionalism for programmers, is described in this chapter.

A. Professional Motivations for Open Source Participation: Transparent Systems[65]

Many interviewed members of both groups explained that an important motivation for their interest in open source was the 'transparency' of source code within the project: unlike proprietary software programming, any interested individual with sufficient technical skills can look at the source code and understand how the software functions, and if interested can improve the code or extend its functionality. This allows members the ability to look at and work with the source code of a program that interests them which was crafted by other skilled programmers. A member of Star Linux explained that this transparency of the source code is an important motivation for his participation in open source (italics mine):

> I believe that there are significant both moral and technical advantages to free software. For someone who thinks like an engineer (and who will hopefully have an engineer title in a few years :), *being able to look under the hood is an essential feature*. Graphical interfaces (most commonly Windows) that neither show you any diagnostic nor allow you to find it on your own, totally repel me these days. I'm very glad to be able to fix problems on my own and contribute the same fixes so that others can benefit from them, too.

Similarly, a senior member of LithiumBSD explained that his frustration with 'black box' source code led him toward the open source movement:

[65] See Adler and Borys, 1996 for an innovative view on bureaucracy, transparent processes and worker empowerment.

> I managed a team that dealt with a whole bunch of Y2K issues and we had a printed circuit board design software package that had been used for years even though we couldn't get support on it, it worked great, and everyone knew how to use it. We discovered that if you opened a file on a machine with a date set past 2000, it corrupted the database. It was really old FORTRAN code. At the end of the day, it mean that we had to walk away from an application that was meeting everyone's needs and go spend millions of dollars on replacement tools all because the lack of source code availability meant that someone else, through their coding ineptitude in this case, but it could be any number of reasons, got to decide when that was no longer useful to us. I found that sort of a hideous state of affairs, so I've tried really hard in my own life to run programs that I have to source code to, and to contribute to projects where the source code was going to be made freely available.

Members of both groups explained that they preferred to use open source software when possible, because access to the source code allowed them to understand and customize the software itself. Another member of LithiumBSD explained his strong preference for the customization available within open source software:

> If I'm going to spend energy today, I want to solve a problem such that I won't see the problem tomorrow. And since I'm going to be working here tomorrow, if I can't actually solve the problem, then a year from now I'm going to be working on the same problem and two years from now I'll be working around the same exact problem. My job doesn't change, I'm already solving the same problem again and again. One of the things for me as a programmer is that I want some variety in what I'm working on. I don't want to be working on the same program all the time because that just gets boring, there's not other way to say it. So the thing I get out of open source is at least the attempt to solve the problem the way that I think I should be solved which may not be the way that IBM thinks it should be solved or Microsoft thinks it should be solved. This preference also comes from my early mainframe days when we had this small number of universities, all of whom were eager to have you work on this source code that they shared.

Typically, 'closed source' programmers are not able to examine the source code of other programming professionals unless they belong to the design team. This approach to programming creates a difficult dilemma for many programmers who find that they are not able to work with 'production level' code until they are hired to do so. This may help

to explain the perennial problem the software industry faces in finding programmers who are properly educated in the process of creating real-world code (Boehm, 2002).

One member of LithiumBSD explained the professional benefits that he gains from his participation in the group that are not available to typical computer science students, or most individuals in the computing industry:

> This is more than a resume entry I have. I've worked on a large-scale project. I've helped coordinate things with multiple people over multiple time zones. I've got an experience with things that I otherwise couldn't have.

Members of both groups emphasized that they participated in open source, at least in part, to get peer review from other programmers that they respect and to improve their programming skills. This motivation for contributing to the group was explained to me by a long-time member of LithiumBSD:

> The main reason that I got involved with LithiumBSD is that when I commit a change to LithiumBSD, of the 500 members, there are maybe 10 or 15 of those developers who really feel that they own LithiumBSD, that it's their thing and when you write any change to LithiumBSD. They look at your code and they say 'you should do it this way, you should do it that way, the code should look a certain way and this is why. It's not because we hate you. From our experience we have learned that it is better to do it this way than to do it that way.' So you're getting that feedback from people that you respect because of all of the work that they've done. And if you have a good group of people to work with, that feedback is valuable, particularly if you are in a small company, and OSU is sort of a small company, you're not going to interact with other programmers at a programming level.
>
> We have maybe at most ten people at the computer center who actually program. And they're so busy getting they're job done they would never look at your code, they just don't have the time and I don't have the time to look at their code. Some of the members of LithiumBSD, their living is made because LithiumBSD exists, so they really care and they really want the best for LithiumBSD. They want your contribution but they want your contribution to be the best contribution possible, so that's worthwhile in itself.

Programmers contributing to open source projects gain an additional motivation for their efforts by knowing that their efforts will be made available to any interested parties to improve upon, rather than being dependent on a specific organization or funding source. Bruce Perens, a notable programmer both in the BSD and professional programming community provided the following explanation for his long-term commitment to open source in a recent Newsweek interview (Businessweek, 2003):

> I worked for Pixar for 12 years. During those 12 years, every piece of software I wrote, except for one, hit its end of life before I left the company -- the projects were canceled or never deployed. Nothing survives. Now, programmers are like artists. They derive gratification from lots of people using their work. Writing software that just gets put away feels like intellectual masturbation. All of the good comes from someone else participating.

Another senior member of LithiumBSD also explained his belief that a desire to retain the autonomy of source code is an important motivating force behind the open source movement:

> Once we move everybody off the mainframe, we're going to unplug the mainframe and then it doesn't matter how great the idea was that you worked on. You had the most brilliant algorithm, it's gone. Many of the strongest advocates of open source, what they're trying to create is something that can last on it's own, on it's own two feet and not be tied to any particular company.

B. Open Source as an Evolving Professional Community: Transparent Organizations[2]

There appears to be some evidence that participants in open source projects are dissatisfied with traditional methods for managing open source projects and, intentionally or not, are helping develop and alternative occupational model for computer programming. One member of Star Linux provides an assessment of typical industry

management practices that typifies the dissatisfaction that some open source participants have with traditional work environments:

> Most managers don't understand my work or appreciate my skills. The typical model of management is to hire people semi-randomly based on their CV alone because the managers and HR people can't recognize whether someone has the skills or not. Then if someone has no skills they will have to repeatedly break things for at least a year before they are let go, if they can give the impression that they know what they are doing. In contrast, someone who has good technical skills but tries to fix problems [thus 'rocking the boat'] generally won't last a year.

In contrast, a member of LithiumBSD explained that managers and leaders in open source projects gain their status by proving their technical and social abilities:

> The managers of open source projects get to be managers because of the fact that they know more and they actually write codes themselves and it's always better [than the non-managers] and that's how they get to be managers, rather than going to management school.

Many members explained that they highly valued the ability to work on tasks that they enjoy, and most would be willing to take a significant pay cut to work on interesting projects, especially if the projects were open source or had a similar system of managing and organizing software projects. Participants in open source also tend to seek a high level of autonomy from managerial interference in the workplace, which one member described as a preference for managers "who ask questions as much as they dictate orders."

Several interviewees explained that they would prefer to be managed by peers who understand and appreciate their work. In the following statement, a member of LithiumBSD explained that he would prefer a peer supervision system in the workplace with a minimum of formal rules to his current employment situation:

> Most companies can do a lot better by sacking their managers and having the technical people organize themselves. My current manager asked me

> what he should do when employees don't obey his instructions!!! There is no point in having such people. I've only had one other colleague who really understood my code, and a few others who had a partial understanding.
>
> In open source development there is a huge amount of peer review. Both the code standards and the functionality of the binary are reviewed extensively and publicly. For commercial projects there is little review. Most programmers make basic mistakes and don't have the basic skills. To a large extent it's because there are few good people to learn from, but also it's because stupid people are not recognized and are therefore tolerated.

Members of both LithiumBSD and Star Linux tend to believe that their process for 'certifying' that new members are up to par to join the group and mentoring them tends to create new members of the group who share both the values and programming practices of more senior members. As a past leader of LithiumBSD explained, when asked whether it was a fair system to allow individual members to decide whether applicants should be considered as potential members[66]:

> Is it fair? Well you may not think so because the person that you're proving it to is the person making the value judgment. If you disagree with that person, it's unfair. However, the person that you're disagreeing with has proven himself to other people who have been in the project for a while. Good, bad or otherwise we're developing members who are of the same mindset as we once were. Or at least in the general direction that we once were.

New applicants appear to highly value the process of having their work reviewed by other highly skilled programmers through the mentorship and peer review process. Participation in open source groups provides members a way to learn about and program within different parts of the operating system, while a typically programming job would usually require that they specialize within a small portion of the system and often doing

[66] New applicants to LithiumBSD approach a current member and ask them to look over their 'patches' to the group source code. That member then decides when the applicant is ready to be considered for membership. The final decision is made by group leaders who consider the opinion of the mentor and other members when deciding whether an applicant is up to par.

years of 'grunt work' such as 'stress testing' applications. In contrast, a current member of LithiumBSD explained that, while he is 'only a documentation member[67]' with few programming skills, he has gained valuable professional experience from his role in the project:

> For me it's fun. It's related to my job: we use LithiumBSD for part of our infrastructure. Other than being a user of the system, I like working on it because it's a great way for me to understand how a big operating system like BSD does it work, how people do things. I don't think I'm exaggerating if I say that some of the people working on LithiumBSD are some of the best people in the industry.
>
> I think that it's great to be able to learn from them and how stuff works and to be able to go and ask people some questions . . . I think it's a great opportunity to learn something and it's a happy coincidence that the role that I have in the project gives me a good excuse to learn more about the project. It gives me a good excuse to go and bug somebody. "Okay so you committed this change but I don't understand and I want to write up a release note entry for it so why don't you tell me some more." It's kind of nice to be able to do that. That's probably a very different answer than you'll get from anybody else.

Participation within these groups is valuable not only to individuals who are just starting out in their programming careers. Members with years of experience in the industry find they learn best practices for programming from other members as well as different 'styles' of programming. The following member of LithiumBSD explains that despite being a member of the group for several years, he still finds membership in the group to be valuable:

> It shows you other people's ways of doing things, especially in a very large system where you have 500 different people contributing to it, you have 500 different ways of doing things. I've certainly learned a number of interesting programming techniques, things that I would not have come across on my own.

[67] These members are responsible for editing web pages and other project documentation such as user manuals but generally are expected not to change the source code of the operating system.

C. The Professional Ideologies of Open Source Programmers

Many open source developers find programming inherently interesting, which contrasts with the typical attitude of programmers in more traditional software companies. This contrast was pointed out by one member who explained that while many computer programmers merely "work at programming and go home when they're done, *I am* a programmer all the time." Open source projects provide a means for satisfying creative desires in a way that may not be possible in a traditional job. As one member of LithiumBSD explains:

> As soon as I got my first computer in sixth grade, I started playing with it and tinkering with it and modifying it and playing with software. It's just been a progression from that to open source. I love playing with projects. I've always picked up all these little toys and building models or whatever, there's always been some project that I've been working on and this is just the big project that never ends.

Members of open source programming groups appear to be supporting an alternative view of the programming profession than the one currently supported by professional societies which measure professionalism based on a programmers degrees and credentials. For 'hackers' a programmer is only 'certified' if they are highly regarded by their peers. A member of LithiumBSD explained that while in the past it was easier to socialize and 'certify' new programmers in the past when programming teams were forced to work within the same location and provided the following explanation for this phenomena:

> There's a theory that says there are actually few really good hackers after the 80's because in their formative years the way that you learn is from other hackers as an apprenticeship and that occurs inside a setting where you all gather in one place, which was the terminal room. But now everybody has PCs and they're in their own apartments or rooms or whatever so you don't get that same interaction. To a certain extent, that's

> mitigated by the fact that you have IRC and certain other chat protocols but that's not quite the same thing as being in the same room with somebody.

By making the source code to their projects available to the public, participants in these groups hope that this will encourage peer review by other programmers. One core belief that unites the majority of participants in open source groups is an 'ethic of openness,' or the idea that whenever possible useful software and other information should be made publicly available. Many developers explained that they believed that, whenever possible, software should be made available to the software developer community because they believe that in the long term this leads to the creation of better software.

This belief is described in this statement by a member of Star Linux: "I participate because I believe that software should be free whenever possible. When the Internet first started, there was a philosophy that you had to give something back whenever you got something from it." Another developer explains that he's "happier when working on Free Software because I think more people, potentially anyway, can benefit by my work" and a third developer likened open source to professional community where everyone benefited from shared efforts:

> It's the barnbuilding approach. In poor countries, no one can afford to build a barn. You mention the fact that you're going to raise a barn and people come from all around the county and your barn is raised in a day. And in turn you go and help raise other people's barns. That way farmers have a whole bunch of barns set up without having to pay a lot of money to professionals.

In the long run, participants in open source projects believe that they will be able to create better programs than proprietary development projects and that their efforts will benefit the public as well as other computer programmers in general. A member of Star

Linux explains that as a result of this 'ethic of openness', members of open source groups are generally motivated to create better code than paid programmers:

> When I write free software, I know that it's going to be looked at as "this is my baby, it's going to have my name on it." What did I do for a company that is going to have my name on it? Five years from now, who's going to know that I wrote that device that extended that C code? If I put it out there with my name on it, it's my reputation on the line. Chances are that the code that I write in a free software project is going to be lot better than the code that I write under deadline pressure with very little to gain from quality control in proprietary software.

Finally, many developers explained that they participated to help create an operating system that they respect and that is valuable both to themselves and to users of the software. One member explains this motivation:

> They're pouring their energy into this project and they really believe that that's the best thing to do with their time. For instance I have other friends who have poured their time and effort in to some great project at a company and then the company decides we can't really sell this, the market isn't there and the company just throws the project out and that's it. That's it, three years of your life just went out the window because marketing said that they couldn't sell it. If you're a real programming person, a person who enjoys programming, it doesn't matter how much you're paid when they throw it throw the window, it just kills you to see it go. It isn't so much that [participants in open source are] altruistic as that if they are pouring their time and energy into a project, they want to be sure that the project remains available in some way and that there isn't some company that owns it.

In the long run, many developers hope that their contributions to open source projects will help them professionally by expanding their skills or by increasing their reputation as a programmer. While few members of these groups have been able to entirely support themselves by writing open source software, the majority expressed a strong interest in working professionally in a position similar to their role in the open source group they contribute to and most developers work for companies that either

employ or develop open source software to some extent. One of the few developers who has achieved his ideal professional situation describes his work:

> I think I have [my ideal job] right now. I'm working on software that's really, really interesting. I'm working on the cutting edge of research. I work out of my home. I'm working for a small company and it's really a close knit group. I really think that we're making a difference and every bit of code that we write goes out under the GPL (the General Public License used in Linux and many other open source programs). So not only do I get to work on interesting code, I get to work on it under a license of my choosing.

Members of both groups also found that they gained non-technical skills through their open source participation that were valuable to them professionally, such as managing a portion of the project, coordinating their efforts with other members, and effectively providing and accepting peer review of source code. These are the types of 'soft-skills' that traditionally educated programmers typically lack (Conn, 2002). As one member of LithiumBSD explained, new members of the group can find it difficult to adjust to working within a programming team: "There are a lot of lone hackers out there who have written a lot of code and it's a culture shock for someone to tell them that their code isn't good." Participation in open source groups appears to be a low-risk way for new programmers to experience some of the realities of professional programming.

The large amount of projects available within both groups provides any interested members with plenty of efforts to manage within the group. As a result, there are generally many opportunities available for leadership or 'senior member' roles in the project for members with the interest and time available to tackle them. A past leader of Star Linux explained that he actively worked to groom new members for these roles, to ensure that important tasks in the group could be completed by many members:

> One of the things that I did as Group Leader was to talk to Nigel Bishop

> and told him "you know you're not going to be doing this forever and what's going to happen when you're not release manager anymore?" and I got him thinking about it a lot and in the end he put out a call for people to volunteer to be sort of assistant release mangers. He got about a half dozen volunteers and he's been working them pretty hard. There's a mailing list where he and they hang out and he sort of hands out tasks.
>
> He'll say 'here's a list of 20 packages that for one reason or another aren't promoting into testing, you take these three, you take these three, go unravel them and come back and report' and through the discussion that they have afterwards. He sort of leads them to interact with the systems, and shows them how to make decisions about whether is this something where we can wait for the package owner to respond to a bug report or if this the kind of thing where we want to solve this problem and get it into the archive, regardless of what the package maintainers thinks. A lot of that is the classic apprenticeship cases where there are some things that you can't learn other than by hanging out with the person currently doing the job.

The same leader also explained that as a result of the mentoring process in the group, the individuals who are leading the group tend to follow the same practices and pursue similar goals to past members:

> There are these overseers of the project who are always aware of what is going on and that what is done is done correctly, at least in their opinion. What they consider 'correct' is likely to change over time. The project slowly changes direction, but there is enough momentum in what correct is that the persons I was involved with back when I was one of the original 24 members would do it this way and I would say 'please do it this way' so they would move in my direction. That's the way that they learned what was correct when they were young and impressionable. So those people are now a part of those 24 overseers who are now going to be heading north but maybe it's a little west . . .but we still keep going in basically the same direction. It's a little more chaotic in that sometimes there are so many people and sometimes somebody will come in with this really cool idea and everybody will say "Wow, we never though of that." But for the most part, like any large organization, we have a lot of momentum based on past history.

It appears that the open source community offers an alternative professional model for programming that may take hold in the industry if it solves problems that plague the industry such as measuring the skill level of programmers and effectively

coordinating large-scale programming efforts[68]. Intentionally or not, participants in open source projects appear to be contributing to a 'professionalization project' (Brain, 1991) where members of an occupation make use of their shared ideology, expertise and work practices to justify their competing attempt to professionalize the occupation.

If the adoption of open source products increases within business, government, and other organizations, creating the need for programmers to maintain and customize these applications (McMahon, 2003), the ideology of open source groups may gain an important role in defining the programming profession. There appears to be evidence of an increasing trend toward an occupational division of labor within the Information Technology (IT) industry, which includes programmers and systems administrators. In a recent survey by Mahen Tampoe found that personal growth, operational autonomy and task achievement were each cited by 'knowledge workers' as a primary motivation for working in IT by about a third of the respondents (34%, 31% and 28%, respectively), while only 7% cited money as a primary motivator for choosing the industry (Tampoe, 1993). Similarly, a more recent survey found that 'the work itself' and 'opportunities for achievement' were the first two motivators for IT workers, while pay was ranked fourth and job security and working conditions, ranked ninth and tenth out of eleven motivators (Young and Mould, 1994; cited in White, 2001: 316).

While there has been little research on whether the adoption of 'clanlike' structures within software firms improves the software development process, there has been some evidence that horizontal coordination systems, as opposed to 'command and control' structures of management, lead to better performance for software projects (See Nidumolu, 1995 for one example), providing evidence that the open source model of

[68] See Fine, 1984 for a discussion of intra-occupational negotiations for professional status

professionalism may help solve some of the problems that plague software development efforts.

Chapter VIII: Coordination and Role Negotiating within Two Subprojects

The following chapter focuses on two subgroups or 'project teams' within each group, to gain a better understanding of how LithiumBSD and Star Linux function by focusing on the activities of a few members of each group. This section describes the history, systems of coordination, role negotiation and conflict resolution that developed within a subproject within LithiumBSD and Star Linux.

The project selected within Star Linux, the Rules Committee, is an important part of the group's infrastructure. This subgroup is responsible for approving changes to group technical and general policies. In contrast, the subproject selected for observation within LithiumBSD, the Foo Project, is a trailblazing effort to develop a major 'port' for the project. Members of this group attempted to develop a version of the popular FooCompany software for the LithiumBSD system, despite little support from the project and its leaders, and resistance from Foo company lawyers and management.

A. The Star Linux Rules Committee: A 'Transparent' Project[69]

> **Principle 12**: *Time is structured in a distinctly different way from a bureaucracy. In the latter, decisions are made with the expectation of permanence: "This is the right way of doing things." Review processes occur at regular intervals, usually annually in a budgeting process, as a way of checking that things are functioning as they should—but not with the anticipation of making fundamental changes. Thus, structural change in bureaucracy comes as something of a surprise, as a dramatic "break" in the flow of events. A post-bureaucratic system, by contrast, builds in an expectation of constant change, and it therefore attaches time frames to its actions. One element in structuring a process is to determine checkpoints for reviewing progress and for making corrections, and establishing a time period for reevaluating the basic direction and principles of the effort. These time periods are not necessarily keyed to the annual budget*

[69] See Chapter 9, Section C for an explanation of the 'transparent organization' stage of postbureaucratic group growth

> *cycle: they may be much shorter or much longer, depending on the nature of the task. This flexibility of time is essential to adaptiveness because the perception of a problem depends on putting it in the right time frame. If one focuses—as most mangers do—on the issues that must be dealt with in the next week, on will simply never recognize the existence of issues that have a longer "cycle time"; and of course the reverse is equally true. The ability to mange varying time frames is a major advantage of the post-bureaucratic system.*

1) Group History

In the early stage of the Star Linux project, there was no formal committee for setting group policy. The original leader of the group, Donald Wryson, created policy by proposing general policies for the group, which were then discussed and accepted or amended by other members after an extended period of debate. While this method was effective when there were a dozen members, as the group grew into the hundreds of members, Wryson found that he could no longer manage policy discussion for the group which were increasingly becoming large-scale 'flamewars' between him and developers with opposing viewpoints. Later leaders of the group also attempted to manage the policy process, most recently in the case of Richard Sydney, Wryson's replacement, who developed a document outlining the responsibilities of leaders and the process for repealing them. Over time, however, leaders have increasingly 'delegated out' the process of approving policy, to the point where a formal 'Rules Committee' with three members and process for approving policy changes was developed. A long time member of the Star Linux team described the evolution of the Rules Committee:

> I guess it's sort of evolved from a few "how to" documents for a set of tools to a set of best practices and to a set of accepted policies that people can be beat over the head with if need be. This has sort of traced the evolution of the project from a handful of people who sort of believed the

> same things and wanted to work together to do good stuff to now where the project is so big that nobody knows everybody and there are even subprojects that a lot of people don't even know about.

Currently, all changes in the group ranging from minor technical details to the powers and duties of the Group Leader are managed by the Rules Committee and by Sub-Rules Committees responsible for policy within specific technical areas of the project.

2) Group Coordination and Tasks Assignment

Generally, policy is written by policy experts in a given area who debate with other experts through group mailing lists, although any member of the group can join the policy debate. Once a proposed policy change is submitted, it is discussed on the mailing lists, during which time amendments and sponsorships or 'seconds' to the change may be proposed. Finally, if the policy change is supported by the majority of members, Rules Committee members or "editors" add the change to the Star Linux 'rules documents.' A member of the rules committee described the process of negotiating policy changes in the following way:

> The [members of the Rules Committee] are editors in a very narrow sense of the word: we don't actually take (nor would anyone really be allowed to take) our privileges to mean we can veto anything or insert anything into the Policy manuals. Everything that goes into the Policy needs to be done without a vocal and argumented opposition . . . Even opinions voiced by non-developers are taken into account (because any non-developer could be a developer after a while (after passing the NM process), really). We try to judge matters on their merit, not on the sheer number of votes or interest they attract. I can't exactly recall when was it that I last saw an actual vote counting being performed on a policy issue. I do remember one or two cases where two sides were actually confronted in such a way,

and I also remember the issue smelled of holy wars[70].

Several 'Sub-Rules Committees' also exist, often with their own mailing lists and informal leaders. These sub-groups write up rules for contributing to their portion of the source code which include requirements for creating accurate documentation, such as user guides, for their section of the source code, and 'best programming practices' for writing the code itself. Unlike the primary Rules Committee, the subgroups can make policy changes within their 'subpolicy' documents, which then only effect the subproject. The Rules Committee leader emphasized that these groups are trusted to create policy within their area without outside interference from the Rules Committee, and that the Rules Committee is responsible for policy that applies to members as a whole, including member duties, formal powers of positions of responsibility in the group and suggested 'best programming practices.'

Generally, policy is created first by the few technical experts that best understand a given area of the project, and if seen as useful by a majority of members, is then adopted by the project as a whole, after a formal vote. Eitoku, the current Rules Committee leader, emphasized that these groups are trusted to create policy within their area without outside interference from the Rules Committee. He described this approach to policymaking in the following way:

> What the Rules Committee does now is document existing, working mechanisms. New policy is created informally by relevant developers out in the wild, and goes through multiple revisions to get it working right. When things stabilize, they are moved into policy, by this time it is existing practice. This prevents us from putting silly, unworkable things in policy (or is supposed to, at any rate). Also, policy can't be used as a stick to beat the project into compliance.

[70] Debates over technical issues that some 'techies' have a strong and often somewhat irrational or 'religious' opinion regarding such as whether Linux or BSD is a better system. See http://www.catb.org/~esr/jargon/html/H/holy-wars.html for a more detailed definition.

He also explained that subgroup policy is only codified when the current technical experts within an area are no longer available to enforce 'best practices':

> Once a project has reached maturity and the people who started it have drifted out or new members come in, at that point things need to be written in stone so that people who don't have the history and context of the subproject know where to go to create a package that will conform with the rest of Star Linux's guidelines as well as the subproject guidelines.

While changes to group technical policies are handled by acknowledged experts within a designated area of the project who debate the best technical approach, the creation of large-scale policy for the group is more problematic and time consuming. For example, rules that define the powers of the group leader, or the minimum duties of a member, cannot be decided using 'technical benchmarks' and as a result, rules regarding powers and duties are perennially debated issues. Since there are generally no acknowledged experts within these areas, including the group leader, crafting this level of policy is much more difficult and time consuming.

This process of writing the rules that group members are expected to follow will be discussed in the following section. This section will also examine how the current leader and influential members of the group came to be accepted in those roles and generally how policy conflicts are settled within Star Linux.

3) Role Negotiation and Conflict Resolution

The process of appointing the members of the Rules Committee is not a democratic one. The current leader of the group, who was appointed by a past Group

Leader, in turn appointed the other members. A current member of the group described the process of his appointment in the following way:

> Eitoku seemed to have had a revelation one day on IRC that Chris and he needed assistance, so he asked if anyone was interested, 'Nemesis' and I volunteered, and here we are...I should, however, mention that both of us have a history of helping out with policy issues, are involved in a large amount of other discussions in the project and aren't generally considered to be overly clueless :)

As mentioned in the prior section, policy decisions are generally decided by members themselves and even users of the software, rather than the Rules Committee members. Group Leaders also tend to play a marginal role in the creation of policy and they rarely propose changes or engage in technical policy debates. One reason that group leaders have had difficulty leading policy discussions was that other members often interpreted their actions as an attempt to expand the powers of the group leader, or otherwise further his self interest.

The position of 'Rules Committee Manager', which is responsible for leading policy discussion, is currently filled by Eitoku Ming, a senior member of the group who is respected by other members for his diplomacy in handling group conflicts and his knowledge of the project as a whole. He described his formal duties within this position in the following way:

> As the Rules Manager, my general duty is to maintain the Rules documents and the related sub-rules documents [which provide rules for subprojects in the group]. This means watching the discussions on the Rules Committee mailing list, shaping up patches to the document, making sure the formal process for policy changes is followed.

He explained that his formal powers were very limited and that he has "far more power in the Rules Committee as a member of Star Linux (whose packages would be affected by policy changes) than I do as a member of the technical committee." However, many of

Eitoku's most important duties within the group are informal and include the facilitation of constructive policy debates.

The Rules Committee Manager is involved in nearly every policy discussion in some group, and generally his role in these discussion is clarifying other's arguments or points of fact, and moving the policy process along or encouraging further debate. Although he may argue that a proposed change will hurt the group, Eitoku does not propose policy changes himself. His hands-off style of leadership is also evident in his strong belief that experts within a given area should write policy and settle conflicts. This style of leadership contrasts with the more dictatorial leadership of the prior 'Rules Leader.' When the prior leader left the post, and Eitoku was selected in his place, he instituted a more democratic process for policymaking. He described his role in the project in the following way:

> My role has shifted over time. Before I got involved in policy, there was a 'Rules Leader' and only the Rules Leader had control over what went in . . . There was a lot of pressure and invariably, things got delayed, and there were controversial decisions at least some people objected to. Joseph [the prior Policy Leader] quit in a huff due to accusations of impropriety. I knew we needed a mechanism that moves away from heroic efforts of a few key people. So I crafted the 'rules creation' document, moving the rules creation process to a rough consensus building mechanism, and I created the post of editor. Editors have no control over content; that is managed by the membership at large. The problem with this is that when everyone is responsible, no one is responsible -- so lately, I have started taking a more active role by making decisions when we are stymied, and preventing a small vocal group from taking over the process. I now go in and look at the issues under discussion, decide if they have been resolved, and act on it.

A key tactic used by Eitoku in resolving conflicts is to 'sit on' controversial policy changes and other decisions until the group is able to come to a consensus. As discussed in the chapter on group coordination, while the policy process, like

the New Member process, is a simple process which only requires "three co-sponsorships for the change and a member vote," in practice, policy is only made if Eitoku and the Rules Committee team decide that group debate over the issue has continued for a sufficient period of time and that there is sufficient group interest in the proposal. He explains (italics mine):

> The only caveat is that in order to get [a policy change] passed is that you need to convince enough people to vote in your favor." [In one recent case] we spent about six months writing the drafts and now we have a draft that everyone agrees on. Now we just need to send out a general message and say 'this is a Developer Proposal, vote on it' and five other people co-sponsor the change and I have to start counting votes. *What's keeping it at this point is that the people that are pursuing it are busy and as soon as I have time and I know that I'm ready and able, I'll push for a vote.*

Eitoku also fulfills another important informal role within Star Linux by providing summaries of discussion that can span thousands of messages and several years and his wide understanding of Star Linux infrastructure and rules allows him to condense complex technical discussions for other members. He explained that "one of the most crucial aspects of steering the Rules Committee is documenting the consensus that was achieved." This 'consensus building' role is essential for coordinating complex technical discussion that any member or user can participate in.

Through the process of periodically writing these informal summaries of policy debates, Eitoku may also encourage a vote on the issue, if it appears that there is no more useful debate on the issue, or he may mention other issues that have not been adequately resolved through technical discussion. While Eitoku has the formal power to call for votes, he emphasized that he only does so once the 'flamewar has died down' and the major technical issues have been thoroughly discussed.

This hands-off style of leadership parallels the Project Leader's belief that members are able to resolve conflicts themselves and solve technical problems within their area of expertise. The downside of this managerial style is that important decisions may require months or years of debates before being decided, a problem that a Star Linux member described in the following way:

> A few years ago, I was very active on the Star Linux lists. Many issues got discussed. But very, very few of them got ever solved. I could read the discussions from several years ago and many of these issues are still being debated . . . Star Linux has a rules creation process, and that rules creation document [which defines this process] is kind of holy. The Star Linux rules process isn't "change something, try it, if it's good accept the change, if not drop it." The Star Linux policy process is more like "think and discuss an item, repeat, repeat, repeat. If ever a consensus is found, make it a policy item. Live with it for the rest of your life."

A few members recognized that this policymaking approach also allowed the Rules Committee to prevent new policies that were not in their best interest by 'sitting on' the decisions, although they could not think of any specific examples of this occurring. If the Rules Committee was seen as overly conservative in creating new policy by many members, members explained that informal pressure on Rules Committee members or the election of a new Group Leader promising to reform the Rules process would amend the situation.

Another apparent flaw in the Rules process is the face that members apparently can attempt to prolong discussion indefinitely to prevent a vote on an issue, since rules debates are not closed until all major arguments are resolved. One of the members of the Rules Committee explained that while this type of abuse is possible within a process of open discussion with no powerful leaders, and eventually "other members would identify a blatant obstructionist and chime in to flame or outvote them."

When the Star Linux project started, since *de facto* policy was created by the developers doing the programming, there were few policy conflicts and no formal process for creating new policies. As the group has expanded, members have found that it is not always clear how current group rules should apply to specific situations, giving rise to conflicting interpretation of existing rules and the proposal of new policies. A long-time member of Star Linux provided the follow narrative of the growth policy conflicts:

> Very early on, the initial documents that became the Star Linux rules documents were written by the people that were putting the Star Linux packaging system together and they were simultaneously solving the problems of 'what technologies do we need for building the packages and how should it be used?' . . . What started to happen was that as people started putting these tools in use to build packages they'd start to run into cases that weren't documented and they'd say 'okay, how am I supposed to do this.'
>
> There would be some discussion on the mailing list and the result was that the process document would sort of capture the essence of that and add it to the document. It seemed to me that it was pretty late in the group's evolution when somebody said that 'gee, what we have here is a set of best practices or policies that we expect everybody to adhere to. Frankly this change was related to the rise of the New Maintainer process which was an attempt to change the way that we accepted new maintainers and put much more rigor into that. One of the things that came out of that was that we were going to educate every new maintainer about the Rules and the things that you ought to pay attention to and adhere to because if you didn't, things wouldn't work well together. [The current process of making rules] is sort of a 'you have to pay attention and argue against something to keep it from going into policy kind of thing.'

The quote above shows that group policy is generally dictated by the group members who are most interested in the debates and best able to justify their arguments through the Rules Committee mailing list, or as Eitoku explained "project-wide voting is a poor method to decide technical issues." As indicated in the previous section, technical policy is set by acknowledged experts within a given area, who codify 'best practices' for their technical area, after having developed those practices in the process of contributing

to the project. Technical policy conflicts are generally decided by the members that know the issue the best, a process that Eitoku explained in the following way:

> If there is no consensus, the discussion goes on, at some point dies and is brought up again every few weeks or months. Sometimes a single person has enough credibility so he can do something by only putting down a proposal, and people accept it. "Best technical argument" is very difficult: unless you know the 'xyz' package very well, it is not easy to be sure if something is better or worse than some other solution. In cases like that usually the 'package owner' wins . . . So the owner can mostly propose things, and implement them without many problems.

In contrast, project-wide policy often effects all developers and as a result, they all are considered to have a valid voice in the policy making process. These large-scale changes often last for months or years without a definite resolution. Often these debates revolve around 'unresolvable' or 'philosophical' issues such as whether software which is not licensed under a 'free software' license should be included within the project. In this particular debate, the pragmatists argue that the project should include any useful software, regardless of license, while the idealists argue that including commercially licensed software (such as 'closed source' or non-GPL licensed software) conflicts with the philosophy of the project, which is to develop open source software that anyone can examine and improve upon.

Generally, large-scale policy conflicts on the Rules Committee mailing list are settled in one of three ways: either the group tires of the discussion and eventually 'tunes it out,' ignoring any messages that refer to the debate, or the flamewar become a continuos din within the group. Another possible result of these project-wide debates is that a member or members of the group propose an innovative solution to the project, such as new group process or infrastructure that meets many of the conflicting group priorities that fueled the debate. Most recently, this type of solution was evident in the

resolution of a group-wide conflict over the issue of determining whether candidates for membership were up to technical and social par for joining the group. The issue was originally resolved with the development of the New Member process which requires applicants pass technical and 'philosophical' tests before being accepted into the group, as a result standardizing the process, while keeping the time frame for accepting members open, allowing gatekeepers to regulate group size, and ideally quality of new members. As long delays occurred in this new process, a member who further restructured the New Member group to allow it to process applications more quickly was later awarded with Group Leader status.

Since group debates over policy play out within the Rules Committee mailing list, members of Star Linux typically attempt to limit the amount of messages on the list by allowing respected or well-spoken members who advocate a position that they agree with to speak for them. A member of Star Linux explained the existence the 'silent majority' in the following way:

> There definitely is a silent majority which doesn't necessarily voice their opinion but they track issues and occasionally intervene when they feel it is necessary -- for example when they see that those already participating in a debate are not contributing to their cause properly. I have often heard developers say on IRC that they didn't think there was anything they would contribute in discussions where one person was opposed to two or three others because they felt this one person handled their side of the argument just fine on their own. If this was the British parliament, you'd hear the silent majority oohing and aahing and yeaing and naying but in a forum full of technically savvy people such as ourselves, such expressions usually aren't considered necessary . . . Imagine if you had to read what is now a 40-message thread with all the yeas and nays -- on a mailing list the size of the Rules Committee list. It could easily increase tenfold in traffic. The sheer overhead would make it painfully boring :)

The group of members who are more likely to have their opinions respected by other members was described by one member as the 'loud group' within Star Linux.

While it may seem strange that some members feel that their opinion is not heard within an open discussion, members often find that their opinions are buried within the large amount of emails exchanged within the Rules Committee mailing list. This member described this phenomenon in the following way:

> Once a discussion has a dozen or two dozen postings, most people seem to stop reading the thread. Also it gets very very hard to see what is going on. Often there is nobody who sums up a long discussion and shows what the issue is, so late comers can read it and talk about it, too. Also I see many discussions going into the wrong direction. People disagree about one very small and very minor item, and continue to fight about it for a long time.

The Rules Committee members attempt to manage this communicative chaos by restating discussions that may span hundreds of messages within a single email, and by encouraging developers not to 'get off track' by debating minor issues. The Rules Committee, like the New Member process, are member-developed processes for solving infrastructure problems that arise as the post-bureaucratic group grows. As the opening 'principle of the postbureaucratic organization' (Heckscher, 1996) suggests, postbureaucratic organizations are perhaps best governed through an expectation of 'constant change' which necessitates they type of ongoing debate over group rules implemented within Star Linux's 'Rules Committee.'

B. The LithiumBSD Foo Project: A 'Trailblazing' Effort

> **Principle 11**: *The problem of equity acquires some new wrinkles. In a bureaucratic system, the main touchstone is always objectivity and equality of treatment: Thus, there is a constant effort to devise rules for treatment of employees, and to minimize the element of personal judgment. In a post-bureaucratic order there is first of all an effort to reduce rules, and concomitantly and increased pressure to recognize the variety of individual performances. This is a major point of tension in many innovating companies at the moment. It is likely that he solution involves the development of* public standards *of performance, openly discussed and*

often negotiated with individual employees, against which they will be measured; this is an apparently increasingly common approach to compensation.

1) Group History

The 'Foo' Project was pioneered by a senior member of the LithiumBSD group who had been a past Group Leader. Despite his 'senior member' status in the group, he received little funding or other support from the current leaders of the group. He initiated the Foo Project because the software developed by 'FooCompany' was considered to be an increasingly important tool for software development by many Star Linux members, despite its commercial license. FooCompany was apparently resistant to the project, requiring that all members of the Foo Project sign a non-disclosure agreement before viewing the source code, which requires them not to share the 'trade secrets' of the software with developers who had not signed the agreement. As a result of these obstacles, Bernard found that neither FooCompany nor the leaders of LithiumBSD were very supportive of his efforts.

Legal issues, combined with major software development issues, greatly slowed the effort of creating a version of Foo software (also known as a 'port') within a LithiumBSD. Despite these many obstacles, the project has slowly grown from its single pioneering member to a handful of dedicated programmers and a much larger group of users and peripheral members who report bugs within the project and request new features. The long term prospects of the Foo project are unclear, and members of the group are currently struggling to keep the LithiumBSD version of the software up to date with new versions of the Foo software. It appears that these legal and administrative

issues are the types of issues that a ‘skunkworks effort’ such as Foo Project has the most difficulty dealing with, since they lack both the funding and corporate know-how that would be required to hire a legal expert, or to have enough clout to bargain with FooCompany.

The stresses of bargaining with FooCompany, combined with increased work commitments and an ongoing conflict with another developer led Bernard to take a less central role in the project, and eventually a few other developers stepped in to coordinate the project. He described his commitment and breaking away from the group in the following narrative:

> Due to major funding from an outside agency, “all of the sudden it got really important, so Java became very important to me personally. I was mainly working alone and I decided that we should get everyone together to create a shared repository [to combine all of the source code from previous efforts to create the Foo source code] and work together and it worked. As I got more and more involved with realizing, I got my fingers more in the pie and by the next version of the software, I was writing code, although another member fixed several difficult bugs.
>
> Our project got stuck after that version and I got really involved in my project at work, having 100 hour work weeks and I got less involved in the LithiumBSD stuff only because I didn’t have any spare time. I spent about 9 months with my nose in a computer monitor . . . and by that time had a team but I think the team really lost a lot of momentum for whatever reason. I think with any project open source or volunteer, if there's not someone with momentum going, someone to keep people interested in it, it just slowly dies. At that time Len Marshall had a lot of energy and he took a major role coordinating the project.

According to another participant in the project, Marshall’s leadership style was similar to the prior leader:

> Len acted from my perceptive as sort of leader in that he wasn't necessarily doing the work, but he was facilitating work, sort of facilitating collaboration and gather work from other people together as well as doing work himself. He was the first one to put the past releases together and to merge them into a single version. He also played a role in

> the sense that he gave status updates and he seemed to be very knowledgeable about where we are and the project direction which isn't immediately obvious when you try to join Foo Project what's happening and who's doing what, who's keeping an overall view on everything.

2) Group Coordination and Tasks Assignment

The founder of the Foo project, Bernard Rieux, started the project both because it was technically interesting to him and because he needed the software for work. Rather than starting the project from scratch, Rieux located programmers in the Netherlands and Japan who were independently working to solve the same problem, and he used their source code as a starting point for the Foo effort. He found that he was doing the lion's share of the work in the project, and a few years after starting the project, he found that a few other skilled developers, and many 'observers,' who suggested new features and reported bugs they found when testing the software, had joined the effort. This lack of participants in the Foo Project did not bother him, however. He explained:

> To be completely honest I've never had to [motivate other members to participate in projects] . . . It's not been finding people, the problem has been organizing people. Getting people interested either happened or it didn't and for me I've never had an issue of finding people who were interested or even motivating people who are interested . . . it's part of being a volunteer project. Here all I can do is provide the communication saying "this is the Foo project, if you're interested in participating, contact me."
>
> If I'm interested in doing it I'll do it all by myself. It may not go as fast as with somebody helping me but I'm willing to do it all by myself. I did a lot of the stuff all by myself when LithiumBSD stated, so I'm willing to do it all by myself. I don't need you to be interested., that doesn't' make it any more or less fun for me I'm doing it because I enjoy it. However, if you're interested, I'd like you to help me rather than have us work apart .

Rieux made little efforts to motivate others to participate in his project, or to coordinate their efforts when they did so. He believes that if a project is interesting, it

will attract skilled developers, whether or not he attempts to suggest on LithiumBSD mailing lists that others help out with the project. After a few years, the project did gain a few skilled members who were able to make important breakthroughs in the development effort. Rieux also notes that with growth there was an increase in 'hangers on' who were only interested in, although he noted that some of these 'outsiders' provided important bug reports. Restricting participation to the few most skilled developers appears to be necessary when starting a major project that had not been done before[71]. A major participant in the effort described the complexities of their effort in the following way:

> There were separate attempts at porting the software and some of them hadn't been coordinated and some people showed up later with code asking "here's what I've done, can we integrate this [into the source code]?" and others would say, "we've already done that, can we integrate this instead?" Or "the way you're doing it is not the right way." . . . The project in itself has a very high bar for joining. If you want to get involved it requires a significant amount of time just to know the software and to contribute in a meaningful way.

And a similar belief was shared by another major participant in the project:

> In the early stages of the project I think that the perception was that 'we need people that can contribute technically and it was such an early state of the project that the bar was so high that we said "we don't need testers [of the new software] and try not to disturb people who are working." My feeling was that that was appropriate for that time.

To minimize the 'chatter' created by too many 'outsiders' participating in the group, Rieux instituted a formal development group made up of the member of LithiumBSD who had contributed the most to the Foo Project. While this was in part motivated by the legal arrangements he made with the FooCompany, he also found that this allowed the members who 'did the work' to better focus on the project. More

[71] This 'skunkworks' approach to coordinating work mirrors a similar process of growth for the projects

marginal members of the group who were not included in the 'closed group' appeared to respect this decision and one 'outsider' explained that:

> I've never perceived any sort of constraints to get involved. For me it was whether I wanted to spend time on that versus other things. I feel that if I had wanted it, it would have been easy for me to become involved and establish closer relationships with the people there.

Due to frustrations with legal issues, and other conflicts within the group (see the next section for details), Bernard now has taken only a minor role in the project. A few other members have stepped in to take a leadership role within the project, although it is not clear that the project is lead by any one member. The group has also remained small, with only five contributing members at its peak. A member of the Foo Project explained that small work teams and even efforts led by one developer were the norm in LithiumBSD:

> Almost everything at LithiumBSD is driven by one or two people. We may have two hundred committers but the ugly secret is that we only have a few people who are actually doing anything . . . For a project like Java only having one person shepherding it, that's pretty common. In fact when you have two people working on it, that's when you get conflicts.

3) Role Negotiation and Conflict Resolution

Rieux, the founder of the group, negotiated his leadership role in the project both by doing the greatest amount of work, and by handling critical administrative issues such as locating a 'dedicated development server' that users could log into to work on the source code. While typically this source code would be stored within the main LithiumBSD server, FooCompany required that a separate machine be used to store the source code, with its own set of logins and passwords to access the code.

---

themselves. See Chapter IV for more detail on apparent stages of growth for post-bureaucratic projects.

As indicated in the above section, the project gained many ‘onlookers’ who were interested in the progress of the Foo Project, and who made demands upon the more dedicated members of the Foo Project, such as requests for technical support, or peer review for Foo-related source code they wrote. These ‘outsiders’ did not have a major role in the project because they lacked the time or technical expertise to contribute source code. Similarly, one group participant explained how the group leaders earned their positions: “Nobody appoints you. If you’re doing important work in that area, you assume that role.” Technical conflicts within the group were also decided by the members who were currently doing the most work within the project. As member of LithiumBSD who observed the Foo Project for several years noted:

> I think that [technical conflicts were] always decided by the leaders at the time, technical leaders who are the current contributors to the software in terms of time spent and effort. My feeling was that whoever was in that position was going to have the power of decisionmaking . . . Being in that position gave you implicit power over deciding Foo Project’s direction because that's what mostly happens in the open source world. If you’re spending time on something and you're contributing, you effectively become a voice of power.

As the Foo project gained interest from other members, there was also an increased level of technical and interpersonal conflict within the subproject. John Swinnerton, a member of LithiumBSD who was close to several of the founders of the LithiumBSD project, was funded by the leaders of the LithiumBSD project for a long-term position creating a workable version of the Foo Project, which gave him an important technical role in the Foo Project. However, Rieux and other members of the group believed that Swinnerton was not deserving of this position and that his appointment was politically motivated. Rieux believed that Swinnerton was granted this position to irk the primary leader of the group, who did not get along with Swinnerton.

Despite several months of efforts, Swinnerton was not able to produce a working version of the Foo software for the LithiumBSD system and his funding from the outside agency ran out. This conflict came to a head when Swinnerton's funding was cut and he continued to work on the Foo Project without pay. After several months of volunteer work on the Foo Project, Swinnerton was surprised to find that although additional funding was made available to finish the Foo port, he was not consulted regarding which member of LithiumBSD would be funded to fill the technical project lead position, despite his important technical role in the Foo Project in the past. At this point, a flamewar erupted involving Swinnerton and Rieux and other members of LithiumBSD. The original members of the Foo team argued that Swinnerton had done little to advance the Foo effort, and that he should not have been funded in the first place while Swinnerton argued that he was not recognized for important changes he made within a difficult-to-program project. Bernard described his support of the decision to hire Ryan Gray, a new member of LithiumBSD with programming skills related to the Foo project, rather than John:

> I recommended that Ryan be given the contract based on what I've seen of his code, and because he was dealing with some of our developers that were very difficulty to deal with and did it very well and when confronted with sometimes vitriolic and obnoxious emails would respond very quietly and say "let's talk technically here."
>
> John worked for several months, his only job, working full time, did nothing, did not get anything working. He then with great fanfare announced his arrival to the LithiumBSD project. He berated the other developers and publicly claimed how great he was and how awesome he was . . . Not once did you hear anybody on the LithiumBSD project say how great his work was. We agree that his work was great because nobody else was doing it at the time. I wasn't doing it because I was involved in the legal issues and Jim was in the process of moving...but told us how wonderful he was and then got really offended when we didn't recognize how great he was.

While several members of the Foo Project allied against Swinnerton, he argued that this opposition was mainly due to the 'politics' of the situation:

> The reason that I had to resign from the project is because I'm a political player because of my interaction with all three groups [FooCompany, LithiumBSD and another open source group]. It originally was my project, the person who forded it and provided the experience was me. I think that critical leadership roles were filled by me, Bernard and Len.
>
> Not only was I the political leader, I was the technical leader . . . What happened was that the old LithiumBSD guard with their bullshit basically excluded me from the decisionmaking process. They asked the other two engineers without asking me and really didn't do the research and they hired a technical underling to do the work and he basically took credit for my work.

These conflicts were exacerbated by the decision of LithiumBSD leaders to fund a developer other than John to take over the main technical role in the project. An observer of this conflict pointed out that when outside funding sources attempt to fund a developer within a current effort, they often create these sorts of conflicts when the funded developer attempts to take a leadership role in the group which the more established member of the group believe the new developer does not deserve. Since Swinnerton did not go through the regular membership process or prove his commitment to the project, long-time members of the group were reluctant to accept his original appointment to the group and he was perceived as an "outsider" within the group. Personal conflicts with a leader of the group worsened the overall situation, and an observer of this conflict explains that as a result Swinnerton was "excluded from LithiumBSD right from the beginning."

Unfortunately, this conflict was never resolved between the members of the Foo Project. Swinnerton and Rieux both suffered negative consequences as a result of the conflict. After several months of often vitriolic emails, Swinnerton noisily resigned from

the LithiumBSD project and Rieux 'flamed out[72].' from the Foo Project and many of his other efforts in LithiumBSD.

This conflict illustrates the types of difficulties that can occur within post-bureaucratic groups that lack definite positions of project leadership or norms for deciding which member is best suited to lead an effort. By funding Swinnerton without consulting Rieux and other members of the Foo Project regarding the best candidate for funding, the Group Leaders of LithiumBSD circumvented the process of peer review of other members that would have otherwise occurred within the Foo Project in making this decision.

Due to a perennial lack of funding within many open source projects, developers who are well funded often gain influential roles within the group and in turn conflicts arise with long-time members who believe that they should set the direction of the group. An observer of the conflict examined in this chapter described this problem in the following way:

> Companies who invest in developers provide both of these resources, occasionally creating leaders or influential developers that were unknown in the group previously. My perception was that in the way communication happened in this case wasn't properly handled by whoever made the decision [to fund John without consulting Bernard]. As a result, John and Bernard didn't know each other and there was the perception of threat on both ends.

This section of the chapter described the breakdown of the rules of meritocracy within an upstart project in LithiumBSD, both when John Swinnerton was appointed to a long-term funded position within the project by LithiumBSD leaders without consulting Bernard Rieux, the current leader of the group and when, after Swinnerton left that position, when Ryan Gray was appointed in his

[72] Burned out due to becoming too personally involved in group conflicts.

place without consulting him. Both Swinnerton and Rieux perceived these actions as violations of the expectation outlined in the 'principle of postbureaucratic organizations' that headed this chapter that decisions that effect the group be made based on 'publicly negotiated' standards of performance, rather than personal or political reasons (Heckscher, 1996). It appears that s a result of violating these rules of equity the leaders of LithiumBSD have betrayed the trust of two highly active members of the group. Bernard Rieux, once a past leader in the group, no longer feels a strong motivation to contribute to the group and John Swinnerton, a self-described 'key liaison' between Foo Company and LithiumBSD no longer contributes to the project at all.

The conclusion of this dissertation will consider the broader implications of these findings and will provide a tentative theoretical model for postbureaucratic group growth.

Chapter IX: Implications and Conclusion

Introduction:

Organizational theorists have long argued that intellectual or 'organic' labor is best managed using occupational divisions of labor, which place the control over work practices into the hands of workers themselves, rather than administrative divisions of labor which are a better fit for more predictable 'mechanical' labor (Stinchcombe, 1959). The software development industry appears to be especially well suited to management under a craft-base or occupational division of labor, as predicted by the Contingency Theory of organizational development and success. According to contingency theory, work is best managed by workers within less hierarchical groups such as 'clans' (Ouchi, 1980) when there is a high level of task uncertainty, a high level of interdependence among workers, and as unit size increases (See Hall, 1991 for an overview of research on these structural relationships). Large-scale software development efforts often risky and uncertain enterprises with high failure rates (Gibbs, 1994; Glass, 1998), a large number of programmers, and high interdependence among workers who often are simultaneously modifying the same section of the source code and working to ensure that their features and software modules work with other software without creating problems.

This conclusion will argue that the open source movement represents an emerging occupational division of labor for software engineering, where programmers manage software development efforts and a non-traditional 'professionalization project' for programming. The second part of this conclusion will consider the lessons that this study provides in how post-bureaucratic groups function in the absence of formal roles and

systems of conflict negotiation. It appears that as the 'knowledge economy' grows, understanding how post-bureaucratic groups operate will become increasingly important for students of organizations as well as managers of 'complex labor' looking to improve worker motivation and efficiency (Adler, 2001; Smith, 1997).

The final section of this conclusion examines a tentative theory of postbureacratic group growth. This section summarizes several observed relationships between culture, structure, bureaucratization and other variables traditionally measured in more quantitative organizational studies. The section will consider general empirical findings from past studies of bureaucratic organizations and work teams regarding the relationships between general cultural and structural variables and will suggest relationships observed in the two groups studied.

A. The Resurgence of the Craft of Programming

It appears that the open source is an evolving occupational model for organizing the software profession. While administrative divisions of labor, which occur within typical bureaucratic organizations, provide a formal hierarchy of responsibility and decisionmaking, occupational divisions of labor decentralize authority over work decisions allowing workers greater autonomy over their work tasks (Freidson, 1973). John Van Maanen and Stephen Barley suggest that the formation of 'occupational communities' such as the open source movement may represent an attempt to professionalize or 'occupationalize' a labor market and that "the quest for occupational self control provides the special motive for the development of occupational

communities" (Van Maanen and Barley, 1984: 287; Barley and Tolbert, 1991; See also Lawrence, 1998 for a recent ethnographic study of this type of professionalizing project).

Technicians have long held an uncertain organizational and professional position. The occupational and administrative divisions of the software professions have long been noted by observers. J.E. Hebden noted that just as the programming profession was just starting to establish itself, technical workers were divided in the more administratively-minded 'System Administrators' responsible for maintaining existing software and more maverick programmers whose occupational identity 'overflowed' their organizational identity (Hebden, 1976; See also Pettigrew, 1973b).

Non-routine technical labor has long been a challenge for traditional management structures. In an ethnographic study of technical workers, Stephen Barley found that incorporating technicians into existing organizational structures is often problematic because "technician's work caus[es] trouble for vertical forms of organizing precisely because it decouple[s] the authority of position form the authority of expertise." (Barley 1996, 434; See also Barley and Orr, 1997). This organizationally uncertain position places stress on technicians who are treated, variably, as highly valued experts and as service workers by workers and managers who are uncertain of the status position of technicians in the organization. As a result, Barely found that technicians often shifted jobs to new organizations "until they found a situation in which they felt appreciated as experts" (Barley, 1996: 443). Many of the participants in Star Linux and LithiumBSD reported similar managerial and organizational problems in their careers and explained that participating in open source groups allowed them to work for a group where their

efforts were rewarded based on their technical skills rather than defined by 'organizational politics.'

The open source movement appears to be a non-traditional approach to professionalizing or 'occupationalizing' programming[73]. The open source community appears to fit many of the traditional measures of professionalism (Greenwood, 1957) including the existence of a strong professional culture and a professional code of ethics. The ethical component of the open source movement is a commitment to the 'openness' and verifiability of their source code by other programmers. The success of the open source community demonstrates the ability of programmers to design and implement high quality software with a minimal level of managerial oversight. The success of high-profile open source projects such as Linux and the Apache webserver helps to legitimate the ability of programmers to police their own work through a craft model of production. In this way programmers are demonstrating that they are able to supervise their own work outside of the constraints of a typical organization or management structure.

Demonstrating this level of autonomy is an important ingredient in professionalization projects (Hall, 1969). It appears that "software engineers" working on open source projects are making a persuasive argument for occupational autonomy by building a professional culture which the engineering profession as a whole has lacked (Perrucci and Gerstl, 1969; Whalley, 1991) leading to an ambiguous position of the engineering profession between the administrative and occupational worlds (Meiksins and Smith, 1993). The formation of 'occupational communities' can be seen as an attempt to legitimate a competing occupational definition of programming, or in the

[73] The term 'occupationalization' is used to refer to efforts to organize occupational sectors that may not follow traditional models of professionalism. See Barley and Tolbert 1991, p. 7.

words of Van Maanen and Barley (1984: 335) "occupational communities promote self-serving interpretations of the nature and relevance of their work in the organization as a means of generating control over that work." Open source programmers also demonstrate a 'calling' to the profession when they create high quality programs with little or no funding, and release the product of their labors under licenses that release their claims of ownership over the software, rather than attempting to profit directly from their efforts.

A final important element of professionalism that the open source movement has not yet achieved is control over the entry into the occupations through control of 'certification' of professional programmers. Open source groups function as an apprenticeship system for many new programmers who otherwise would not gain experience working on real-world software projects until they started their first real job. If the open source model of occupationalizing programming gains legitimacy, it is possible that personal reputation within the open source community will provide an alternate model for proving programming skills to employers when open source programmers seek jobs.

It appears that by creating an effective process for policing the software occupation and high quality but low priced software, that many open source participants are hoping that their alternative approach to socializing and regulating programmers, along with the software that is produced by this process, will become the new 'gold standard' for programming. The revolutionary open source model of licensing software so that code remains 'open' to any interested programmer to read and modify also insures

that the products of open source labors remain in the hands of the programmers themselves rather than becoming the property of a specific individual or organization[74].

This emerging craft model of software professionalization may gain popularity if currently accepted models of managing programmers continue to fuel large-scale software failures (McBreen, 2002; Gibbs, 1994; See Glass, 1998 for a description of the most egregious failures in the last few years). The limited adoption of coalitions with open source projects and open source management models by many large software firms including Sun and IBM provides evidence that the open source approach to occupationalizing programming may be gaining legitimacy within the software industry.

Market trends such as global outsourcing of programming efforts, the rise of 'network organizations' and work teams (Smith, 1997; Castells, 1997), the widespread failure of many large-scale software development efforts and attempts to flatten the hierarchy of organizations through matrix model and other organizational restructuring efforts may further increase the popularity of the decentered open source model of managing programmers. In *The New World of Work*, Stephen Barley explains that this decentered model of management has become increasingly widespread as managers in may industries have "f[ound] themselves relegated to the important but less heady role of coordination" as "domain specific" skills which are "too complex to be nested" rise in importance within an industry (Barley & Orr, 1997:4).

The organizational structure and roles that evolved to meet these challenges is the focus of the second section of this conclusion which reconsiders the culture, roles and

[74] The fact that BSD's licensing scheme provides a much weaker protection for this openness—namely by allowing companies to use their code without keeping their products open to other programmers—may explain in part the lower participation rate in BSD projects compared to Linux distributions.

systems of conflict management that evolved within the two open source groups examined in this study.

B. The Work Practices and Culture of Two Post-Bureaucratic Organizations

The second major objective of this dissertation was to conduct an ethnographic study of the culture and work practices of post-bureaucratic organizations. A similar study of the shift from a traditional bureaucratic organization to a matrix structure was conducted by Calvin Morrill who found that as a traditional bureaucratic firm shifted to a matrix structure, challenging norms of status and conflict management, managers evolved a new system of roles and ceremonies to resolve role uncertainties. This dissertation has attempted to gain a similar understanding of the processes and culture that evolve in open source groups in the absence of bureaucratic systems of managing work and defining work roles.

One lesson from this dissertation which is important for organizations attempting to develop 'worker empowerment' or 'team work' reforms by creating less hierarchical structures is that workers will only commit to these reforms to the extent that rules, work processes and important decisions are made by workers themselves within a system that rewards members for their technical and organizational contributions to the group. While Star Linux excelled in this form of trust building by opening up every part of the organization and decisionmaking to members including the creation of group policies, the election of leaders and the statement of the groups 'philosophy,' LithiumBSD has retained a core group of leaders who are only partially accountable to group members. The major gap between participation levels in these two groups, evinced by the

exponential growth of participation in Star Linux, appears at least in part due to these differences in organizational openness.

In the groups examined in this study, individuals holding formal administrative roles had very little decisionmaking power and several leaders explained that they rarely had more influence in decisionmaking than other members. Rather, their greatest responsibilities involved motivating other developers to complete important projects within the group and developing infrastructure to help track project efforts and simplify work procedures. Similarly, current members emphasized that seniority had little relation to individual influence within the group and pointed out many cases where a new member of the group quickly became influential when they proved their technical value to the group.

These findings are in keeping with current theories of trust-based organizations. In Paul Adler's study of the varying economic performance of trust-based organizations, he found that these groups were most effective when they used 'reflective trust' which allowed workers to question the basis of any organizational practices and suggest reforms as needed[75]. He explains that this type of trust is akin to the open and relatively status blind process of deliberation described in Jurgen Habermas's model of civil society (Habermas, 1989), however at the organizational level and the legitimacy of reflective trust "is derived from grounding in open dialogue among peers" (Adler, 2001:227). In order to build trust within their organizations, it appears that managers must allow workers to gain both a broad understanding of the rationale behind work practices and principles as well as the confidence to question and modify them as need. It appears that

[75] This is compared to the 'unreflective trust' of pre-modern clans where 'believers' were expected to accept organizational goals without question based on traditional beliefs.

the extent to which managers accept the non-traditional and 'less heady' role of coordinator and motivator of employee efforts is the extent to which they will benefit from the effects of 'reflective loyalty' to the organization, as long as they ensure that employees can be trusted with the responsibility of self-management.

Adler's study of trust-based organizations provides a second important principle for worker-led or 'occupationally oriented' organizations such as Star Linux and LithiumBSD: trust-based organizations are most effective when the culture and practices developed by workers are balanced against the pressures of either the market or hierarchy. It appears that the move of the open source world into both the commercial marketplace and the involvement of more hierarchical organizations such as IBM and Sun in directing open source efforts will exert positive effects on the open source movement by checking some of the more pernicious tendencies of a craft-based model of software production, including a lack of focus on user needs and marketable products.

C. A General Model of Post-Bureaucratic Group Growth

> Things were much easier when there were only 20 different committers, but the very nature of the project has to change as that group grows to 300 committers. LithiumBSD has been trying to ignore that fact for the past year or two, and we're now forced to realize that we really have to have *some* management-like structure just to remain productive.
>
> --Long-time LithiumBSD member

Both LithiumBSD and Star Linux followed a similar pattern of growth, which may provide a general model for the successful expansion of post-bureaucratic or clan-type organizations. Unlike traditional organizations which are able to codify standards for

organizational roles and responsibilities, allowing for a relatively painless transition with growth as new organizational members move into well-defined roles with codified policies and responsibilities (Weber, 1922). Post-bureaucratic organizations require that new members internalize group principles before they can be accepted into the group and gain important positions (Ouchi, 1980). Members of post-bureaucratic groups must demonstrate that they understand and respect these principles in order to build enough trust within the group to be allowed to self-manage their contributions. Generally there are little or no formal enforcement systems within these organizations to ensure that workers adhere to these rules.

As these groups grow, building the level of trust required for self-management of work becomes problematic. Unlike bureaucratic groups which can codify organizational practices and responsibilities, post-bureaucratic groups can only function if the majority of their members understand the principles which organize work and have internalized group culture to the point that they can self-manage tasks and negotiate their position within the organization. Based on group histories provided by individual members, several of whom had been members of the groups from the beginning, I provide a tentative model below which summarizes the process of rapid growth within LithiumBSD and Star Linux and the changing organizational forms that arose as the groups grew.

Organizational theorists have long been interested in measuring the relationship between cultural and structural variables such as the level of employee commitment to organizational values and worker efficiency. Past studies of bureaucratic organizations provide findings that only partially agree with the findings of this dissertation and the

model of postbureaucratic group growth provided below. Three relationships that were generally not verified by this study include an increase in bureaucratization, centralization and complexity within organizations over time (See Hall, 1991 for an overview of research on these relationships). In Star Linux and LithiumBSD, there was an expected increase in consensus-based rules and processes however this was not a typical process of bureaucratization because members could generally adopt or ignore the rules which functioned as suggestions or 'best practices.' While bureaucratic organizations tend to centralize decisionmaking power as they grow, Star Linux and LithiumBSD actually became more decentralized with growth as group leaders increasingly delegated decisionmaking power[76]. Finally, there were mixed results regarding changes in organizational complexity with growth. Star Linux and LithiumBSD exhibited an increase in horizontal differentiation, which is measured by an increase in specialized roles and projects[77] (Blau and Schonherr, 1971). However, the level of vertical differentiation, measured as the number of authority levels in the organization (Hall, Hass and Johnson, 1967), actually decreased somewhat leaders gave up many of their formal powers as the groups grew creating a relatively meritocratic craft-based structure for organizing work (Stinchcombe, 1959).

Other traditional findings of organizational theory appear to have been verified by this study including an expected positive relationships between the strength of group culture and organizational performance (Gordon and DiTomaso, 1992; Denison, 1990; Collins, 2001) and a prediction that as 'task uncertainty' increases, groups will adopt group-based, rather than top-down forms of managing work (Van de Ven, et. al., 1976)

[76] See the three stage model of growth outlined below

and craft-based divisions of labor. The 'stages of growth' outlined below provide some tentative explanations for these observed relationships.

It is especially difficult to determine whether a culture distinct to each group studied effected the structure and performance of Star Linux and LithiumBSD due to the small sample size of this study. However, the majority of modern organizational studies do not consider the content of the culture under study and rather focus on the intensity of workers belief in the shared culture[78] and members of both groups evinced a high level of belief in group values. These tentative findings can be better verified through a broader and more quantitatively-oriented study of the hundreds of large open source projects in existence, or studies of emerging postbureaucratic groups and work teams in other professions, especially within emerging technical occupations. A general model of the stages the groups moved through over time is provided below as a tentative model of postbureaucratic group growth.

Stage 1, "Trailblazing" (Estimated size: 5-25 members):

Both groups were founded by a few committed and skilled programmers who were able to devote a great deal of time to their 'labor of love.' In both groups, a single leader managed nearly all aspects of the project and was responsible for all major decisions, and generally the founders were content to limit participation to a few committed members. At this early stage, few systems of tracking work and coordination of efforts were created within the groups and work was coordinated through frequent

[77] See the discussion of the Rules Committee and Foo Project in Chapter 8 for an example of two of these projects
[78] See Martin, et. al., 1998 for a recent exception

communication among all members, by email or in person. In both groups, there was a period of slow growth in membership with as few as two new members added each year. A founding member of LithiumBSD provided the following description of the group's beginnings as a small group of programmers devoted to 'getting it right':

> LithiumBSD tended to have people that were a little bit older, people who were professionals. I was still in school but I had been working as a professional programmer for a number of years. I'll just give a little bit of the culture: I think that was what drove us towards LithiumBSD more than anything else was we had seen how things were done in the commercial world "just get it done, don't get it right." We were really intrigued and challenged by the LithiumBSD effort. We had a lot of energy and liked what we were doing and we could do it right. We believed we could do things the way we always wanted to do it at work: don't rush, analyze the project, take the time to determine how best to do it, analyze data structures, make it clean, and rewrite it if necessary. We weren't trying to compete with anybody at the time, but we wanted something that was commercial quality and that I could actually use it for my day job and at night I could hack on [program for] LithiumBSD.

The groups did not attempt to formalize socialization into the group since new members tended to already have the technical skills and commitment to the group required to manage their own work. Rather, members were able to coordinate work successfully and solve problems because, as founders of the project, they already had a deep understanding of the group culture and coordination processes. A current leader in LithiumBSD explained how a culture of self-management functions within the group during this stage:

> In the original days 12-24 members were really active and all of us knew each other. It was pretty tight knit and when you did something wrong, there was the culture of BSD. Did you hear about the term 'pointy hat'? In LithiumBSD we have the 'pointy hat'. When you screw up, you 'sit in the corner with the dunce cap'. So I would say 'I get the pointy cap, I'm the dunce, I screwed up today.'
>
> All of us had access to all of the code and everybody trusted everybody. I did not necessarily know where you live but I had a relationship with you, inasmuch as I had a relationship with anybody who

> I deal with every day and I trust you, because you've proven yourself. The reason that you are involved in this is that back in the early days you were here even when it wasn't sexy to be a part of LithiumBSD. It was just something to spend a lot of time with and it was something that we all felt passionate about. There were no kudos. We were just doing it because we enjoyed it. There was some public, in that a few people had hear of it, but it wasn't well known, so with that came a lot of trust.

At this stage, both LithiumBSD and Star Linux were led by a single, highly-skilled leader who could oversee all parts of the project. As the groups slowly expanded their membership base, other members with a high technical skill level and competing views regarding the technical direction the project should take joined the group and conflicts began to occur as developers negotiated new roles within the group. The same respondent describes the effect of these conflicts between senior developers:

> To tweak something from scratch takes a lot of ego and an attitude of "you may not agree with the way that I do things but I'm going to get it done." So there were a lot of egos and as the project grew, a lot of the egos got bigger and a lot of people left when people didn't agree with the direction the project was going, and this is what eventually happened to me. I was a leader and I resigned.
>
> The leaders became the guys who were the most active and the other guys weren't. It just sort of happened that way that they were the most involved. Eventually these egos clashed. I couldn't get along with Michael, his lack of direction. That was my opinion at the time was that he had this title of Architect and he never did anything with it. I berated him for having this title and not doing anything about it, and that was his personality. If you're going to have a title, live up to your title. I got tired of it and he wouldn't make decisions, he was always getting feedback and similar conflicts happened with major developers. Again, we had to have big egos to get the project started.

As both groups grew, leaders found that they could no longer supervise all the contributions to the group and that they needed to start codifying organizational practices and delegating some degree of managerial responsibility. A long time member of Star Linux describes this process:

> There were about 50 people on the mailing list at that time and everybody contributed what their opinions were but the final authority was Ray Johnson. His role was closer to what Linus's role has been in Linux, he was the gatekeeper. He was the only person that had the authority to do anything and he made all the decisions and he rapidly approached burnout. After about a year of this he found that he had spent too much time, that the distribution had begun to lag behind the upstream sources [external programs 'ported' into LithiumBSD] since he didn't have the time to upgrade everything. He couldn't package everything and take part in all of the decisions and where they were going. This was the first crisis point, where he kind of flamed out and Mark Beauvioux and Darpak and a whole bunch of people jumped into the breach an this was the first time I heard speculation that Star Linux had been getting too big and there was no way it would survive because Ray Johnson was getting burned out.
>
> He also found out that leading Star Linux was kind of like herding cats, you found out where the cats wanted to go and then herd them. And he was not interested in where we wanted to go. He said that we had grown far beyond a small group where everybody know everybody else enough to trust their judgment with the code and he decided that in order to survive we needed to have a set of formal rules and roles and he drafted them. That was the significant change in how Star Linux operated because now we had a well-designed set of rules and that's when the project leader position was formalized.

As the central leader in each of these groups found that he could no longer effectively direct the project, the groups moved into 'Stage 2' of growth by developing coordiation systems and delegating managerial reponsibilities to other members to distribute the coordination load that was handled by a single leader.

Stage 2, "Passing the Torch" (Estimated Size: 25 – 300 members)

In this stage, the leaders and senior members of the groups started to develop formal and informal coordination systems which could decentralize management by automating important processes or delegating them to trusted members of the group. One major change at this stage for both groups was the development of a process for ensuring trust by testing whether new developers possess requisite technical skills and

commitment to the group goals before they are allowed to join, as well as the development of a formal mentorship process for new members. Several senior members of Star Linux created a formal exam to test new candidates for membership: applicants are asked both technical questions related to group practices and about their knowledge of the group philosophy. LithiumBSD, in turn, implemented a less formal system, which requires that before being admitted as a member, applicants submit dozens of bug reports with source code that fixes them or "patches" to a current member of the group who can then choose to recommend the prospective member to the leaders of LithiumBSD or may ask for more patches from the applicant.

When the groups were initiated, the members generally had a uniformly high level of technical skills and as the groups have expanded, both groups accepted members of varying skill levels. Rather than requiring that new members be highly skilled programmers, both groups have developed levels of membership with varying levels of responsibility and skill requirements. For example, both groups have created members who are responsible for modifying project documentation such as help manuals and group web pages. While these "documentation members" technically can modify any part of the source code, there is a strong informal pressure against them doing so. These lower-responsibility levels of membership allow the groups to accept members who have not yet proven that they can be trusted enough to modify the project source code. A member of LithiumBSD describes how membership standards were lowered as the group grew and how conflicts became increasingly common as more 'unknown elements' were accepted into the group:

> Sometime about 2000 or 2001, the number of committers in the project grew explosively. We decide that rather than being harassed about letting

> people in, the only thing that we needed for membership was some evidence of an ability to program and a willingness to contribute to the project and on a not very strong say-so we started letting people into the project. We pumped up from about maybe 60 developers to somewhere in the vicinity of 200 in the space of maybe a year and a half. We were adding new members almost daily.
>
> When you grow explosively like that, the finite chance exists and in fact approaches unity that you're going to start getting some trouble cases. That happened sort of and at that stage we started seeing some disunity in the project where person X started being a lot of trouble and person Y turned out to be a bit of a fighter and issues like that.

Another important development in this stage of growth is the articulation of group principles and values to better socialize group members such as a 'statement of group philosophy' as well as the codification of technical processes and the development of formal and informal coordination systems[79]. For both LithiumBSD and Star Linux, at this stage of growth these 'statements of belief' and summaries of best practices were written by a central leader who spent several months collecting feedback from group as a whole to ensure that these statements of principles were supported by the majority of the group. A major infrastructure development in both groups was the creation of a formal process for accepting new members. Star Linux created a highly formalized process for certifying new members which requires that applicants verify their identity through other members, pass a test on the technical rules of the group, and answer several questions regarding the philosophy of the group[80]. However, there is evidence of resistance in both groups against formalization of important group processes. For example, despite attempts at standardization, the New Membership process can only be managed by highly-trusted members of the group who can be trusted to decide which applicants should be granted the responsibility of modifying project source code. Despite attempts to codify this

[79] See Chapter V for more detail on coordination within the two groups.
[80] The New Membership process in both groups is detailed in Chapter V

process in Star Linux, a senior member of the group describes how the members trusted with managing the new membership process were overzealous in their role as gatekeepers for new members:

> At the beginning, members were more likely to be coding professional and other people using the packages, nowadays especially after the New Member process, there is a perception that this group of Star Linux members and the backgrounds that they come from has widened considerably form the days of yore. I'm not going to offer an opinion about whether this is a good or bad thing.
>
> As we grew, the New Member policy was developed. After that, two people who were handling the New Member process grew very dissatisfied with either the applicants that were coming in or the work that they had to do, it wasn't clear because they didn't say what was bothering them, they just stopped letting new people into Star Linux. For 18 months or so there was nobody getting in.

In contrast, LithiumBSD relies on the judgment of individual members in deciding when a new member should be accepted into the group and their decision is then finalized or denied by the group leaders. At this stage of growth, LithiumBSD instituted a minimum period of mentorship for new members before allowing them to make unsupervised changes to the source code. This tendency towards preferring informal decisions made by senior developers over standardized practices and roles is also true of the LithiumBSD project as a whole when compared to Star Linux.

LithiumBSD tends to have a more hierarchical organizational structure and greater decisionmaking power concentrated in the hands of senior developers. In part, this difference appears to be due to the higher level of trust required to join the group. A member of LithimBSD who was a member of the group from the beginning provided the following narrative explaining the growth of the group into 'Stage 2':

> They formed this core team which was basically Macario and his buddies. No one knew what they did. No one knew when they deliberated. People would go to them with deal and offer them stuff like two big SPARC

> systems which at that time cost a lot of money and on one knew they got those and they kept them. I asked them "Can I do a port to one of these things" and they said "well you have to be an employee" [of the open source company where the leader worked] and to this day they have not done anything with those. So those guys were a very closed organization.
>
> This led to a developer revolt as we got bigger. When there was only 10 or 20 of us, this led to a developer revolt. People were very unhappy with the leaders, which is when we had public leader elections, whereas before what happened was that Macario decided "okay I like you, you're in the good old boy network, you're a leader." People were dissatisfied with that because they thought that some of those leaders didn't do anything but they had to conform with the edicts that they were passing down and they had no say in what the leaders were mandating because they weren't leaders.
>
> So this led to a developer revolt and this led to open elections. But even then, Macario did a smart and savvy thing and he said "okay we can have elections but I don't want this to turn into a political thing where we have debates and all that campaigning. People should vote for people based on their mailing list posts or contribution to the group or whatever." The criteria were never clear. So in the first election there was no campaigning and no one knew about anyone but the current leaders. So guess what? Half the original core members that ran got reelected again so it was more of the same. The current leaders still don't do anything consequential.

Possible causes of these differences in organizational structure and group culture in the two open source groups will be considered in the following section.

Stage 3 "Transparent Organization" (Estimated size: 300-5000+ members):

The final stage of post-bureaucratic group growth requires formalizing group policy and fully decentralizing decisionmaking and setting of technical vision into the hands of all group members, and to a certain extent, to the users of the software who are able to contribute to technical discussions on the group mailing lists. This stage of growth in the Star Linux project is described by a senior member in the following narrative:

> We are growing. It all started as a one-man show, with a few cheerleaders. Then the cheerleaders started contributing (about where I joined). We are now beyond the level where everyone knows, and trusts, everyone else.

> We have had to evolve, and adjust to growth. The New Member process is a result of that. Developing a Rules Committee so no one person is critical in developing policy is a result of that. Communication is far more critical now than it was before and we also need objective rules to allow the group to continue to cooperate and a thousand people are far harder to move to a goal than 50.
>
> Unfortunately, despite the philosophy questions in the New Member process [which tests member knowledge of group goals], the project has grown too large to have a single goal, which is a good thing: we can now explore all kinds of directions, and let selection pressure determine which ones are viable.

In this stage there is an increased development of grassroots administrative structures although most rules are enforced informally either through self-enforcement or by peers. Generally, while formal bodies exist for conflict resolution and policymaking[81] they are rarely used and developers are expected to resolve conflicts informally, and to develop group rules which best fit their work processes when rules are not otherwise specified.

Despite the large size of the groups in this stage, there is a high level of consensus regarding important group decisions, even if consensus building requires months or years of group discussion. As a result, there is a very conservative attitude towards modifying group policies and making large changes in the group. This conservatism in formal rulemaking does not greatly stifle the actions of the group as a whole, however, because developers are allowed a great deal of freedom in deciding how to approach group problems and when to start new projects within the group. New practices and projects that develop then have the potential of becoming the new 'best practices' for the group if enough members start to use them.

The finding that these groups actually grew increasingly decentralized as group size increased conflicts with past findings of organizational growth which generally find

that increased group growth leads to both increased formalization of rules and an increased centralization of decisionmaking power. One possible explanation for this finding can be drawn from Paul Adler and Bryan Bory's study of bureaucracy in organizations (1996) which examined the conflicting results of past research measuring the relationship between level of bureaucratization and organizational efficiency and survival. Adler and Borys suggest that the *type* of bureaucracy determines whether employees will consider organizational rules to be more empowering rather than the *extent* of bureaucratization. Rules created under the logic of 'enabling' bureaucracy incorporate workers' perspective into rulemaking in order to improve work processes while 'coercive' bureaucracy is generated almost entirely top-down by managers who have other high-level concerns in mind. The high level of enabling bureaucracy in LithiumBSD and Star Linux, combined with member-controlled processes for making organizational and work process rules, helps ensure that decisionmaking power generally remains in the hands of members themselves and that the groups are not overly hindered by the relatively large number of rules and coordinating systems within the groups.

In this stage of growth there is also a high level of consensus supporting the legitimacy of the group leaders. The developer body as a whole votes leaders in, after a period of presenting a platform or debating other candidates to ensure that members have a fair idea of the direction that the prospective leaders intend to take the group. Other leadership roles and managerial roles within the project are formally defined (even if mostly self-selected) and there is a policy document declaring the extent of a leader's powers and a formal process for the project members to overrule them.

[81] See Chapter VI for detail on the process of crafting group policy in both groups

Two important organizational developments at this stage are an increasing respect for non-technical leaders who possess administrative skills and are able to develop improved systems of communication, coordination and problem solving within the group, and the evolution of leaders have few powers that are not available to the typical member. Rather, at this stage leaders tend to be effective networkers who are able to identify and encourage other members to fulfill needed roles in the project, and members with a high level of administrative skill in developing necessary processes to support member efforts. This style of leadership requires a broad vision of the needs of the project as a whole and the belief that specific technical experts can be trusted to solve problems that crop up within their chosen domain.

This leadership trend appears to be evident in Star Linux with the election of a project leader with little or no programming skills who proved his value to the project through his willingness to solve organizational bottlenecks by first doing much of the needed work himself and by delegating new positions in the group as needed. Similarly, at this stage there is a further lowering of the bar for membership by formalizing the standards for members that are responsible for webpages or documentation and other administrative functions, and an increased respect within the group for individuals willing to fulfill these roles.

It appears that LithiumBSD has not reached this third stage of growth, in part due to the higher level of skill required to be a 'kernel hacker' (a programmer who has the ability to modify the core of the operating system itself—an ability available to any LithiumBSD member) than to simply be a 'maintainer' who is only responsible for ensuring that external programs such as text editors work within the operating system

(the highest level of responsibility available to members of Star Linux). Members of the Star Linux project are limited to this level of responsibility because they cannot modify the Linux kernel, which is developed by the Linux project.

As a result, it is more difficult to determine whether candidates for joining the LithiumBSD project have the requisite level of skill and commitment to deserve this responsibility than it is for Star Linux to expand their membership base. It is likely this difference has slowed the growth of the LithiumBSD team when compared to Star Linux. An additional piece of evidence for this hypothesis is the fact that LithiumBSD members tend to disvalue members who possess administrative or managerial skills unless they also have a high level of technical skills. As a senior member of LithiumBSD explains:

> The culture of this group is such that technical skill is required for credibility in other areas. This is a problem, as there is often no correlation or an inverse correlation between technical prowess and administrative or leadership or communication skills.

The differences between the Star Linux and LithiumBSD regarding several important categories or group infrastructure and decisionmaking are summarized in Diagram 1 below. As described above, the primary reason for the differences appears to be the difference in the type of work that typical members are involved in. Namely, members of LithiumBSD tend to be involved in more complex work than members of Star Linux because LithiumBSD members program the kernel of the operating system itself rather than adding programs to an existing kernel. This more complex level of work requires that a higher level of trust be obtained by LithiumBSD members before they become full members and gain positions of influence in the group.

DIAGRAM 5: CHARACTERISTICS AND STAGES OF GROWTH FOR GROUPS STUDIED

	LithiumBSD	**Star Linux**
New Membership Process	Informal decisions by individual members. Leaders review member decisions to accept new members.	A standardized process with well-defined stages but 'gatekeepers' can prevent group growth.
Policymaking and Rule Enforcement	Generally top-down rulemaking and enforcement informed by best practices of the group.	A highly consensual process which can be very slow; any member can suggest policy. Reflects best practices but difficult to revise.
Conflict Resolution	Primarily settled between developers. A formal but mostly inactive body exists for resolving technical conflicts.	Primarily settled between developers. A formal but mostly inactive body exists for resolving technical conflicts.
Stage of Growth	Stage 2 with some characteristics of Stage 3	Primarily Stage 3

Appendices:

A. Principles of the Postburecractic Ideal Type, from Charles Heckschler, 1994.

Principle 1: "In bureaucracies, consensus of a kind is created through acquiescence to authority, rules or traditions. In the post-bureaucratic form it is created through institutionalized dialogue."

Principle 2: "Dialogue is defined by the use of influence rather than power: That is, people affect decisions based on their ability to persuade rather than their ability to command. The ability to persuade is based on a number of factors, including knowledge of the issue, commitment to shared goals, and proven past effectiveness. It is not, however, based significantly on official position. Relations of influence can and do form a hierarchy: some people are more persuasive than others. Thus this system is not in any strict sense "egalitarian." But the influence hierarchy is not embedded in permanent offices, and is to a far greater degree than bureaucracy based on the consent of, and the perceptions of, other members of the organization.

Principle 3: Influence depends initially on trust—on the belief by all members that others are seeking mutual benefit rather than maximizing personal gain. Without this basic trust, persuasion is impossible, because everyone assumes that others are trying to "put one over" on them. A system stressing influence must have a higher level of internal trust than one based on command and power. TH major source of this kind of trust is interdependence: and understanding that the fortunes of all depend on combining the performances of all. Specifically, in a business, it derives from an understanding of the ways in which different pats of the organization contribute to the accomplishment of the overall strategy.

Principle 4: Because interdependence around strategy is the key integrator, there is a strong emphasis on organizational *mission.* The trend has been to focus on how the company actually seeks to achieve rather than on universalistic statements of values. It is often hard for an outsider to understand what all the effort is about: Most mission statements seem remarkably innocuous. They say something about improving quaitl9yu and cutting costs, and something about the kind of a business the company is in—rarely anything surprising. But the mission plays a crucial integrating role in an organization that relies less heavily on job definitions and rules. Employees need to understand the key objective in depth in order to coordinate their actions intelligently "on the fly."

Principle 5: In order to link individual contribution to the mission, there is a wide-spread sharing of information about corporate strategy, and an attempt to make conscious the connection between individual jobs and the mission of the whole. This enables individuals to break free of the boundaries of their "defined" jobs and the think creatively and cooperatively about improvements in performance. In the past decade, companies have greatly increased the dissemination of information about the systemwide

performance, even to blue-collar employees. A decade ago, very few companies gave productively data to blue-collar employees; now it is common. And even at the preset day, in my own research, I have found most middle managers struggling to cope with the flood of new information about strategy and mission that they have been asked to absorb. Information technology has greatly facilitated the dissemination of information. Since computer networks tend to be quite open, this information often flows not just from the top down but is criticized and add to from the bottom up. This process increases the *credibility* of the data being shared.

Principle 6: The focus on mission must be supplemented by guidelines for action: these, however, take the form of *principles* rather than rules. The difference is that principles are more abstract, expressing the *reasons behind* the rules that are typical of bureaucracy. The use of principles carries a major advantage and a major danger. The advantage is that principles allow for flexibility and intelligent response to changing circumstances: People are asked to think about the reasons for constructing on their actions, rather than rigidly following procedures. The danger is that this flexibility can be intentionally or unintentionally abused, threatening the integration of the system. The dangers are reduced by two mechanisms: the creation of trust, derived especially from a clear understanding of the interdependence among all; and by periodic review and discussions of the principles to be sure that they accurately capture what is needed and have not been distorted. Post-bureaucratic organizations spend a great deal of time developing and reviewing principles of action.

Principle 7: Because of the fluidity of influence relations by comparison to offices and authority, decision-making processes must be frequently reconstructed; they cannot be directly "read" from an organization chart. The choice of "who to go to: is determined by the nature of the problem, not by the positions of those initially raising it. Thus, processes are needed for *deciding how to decide*—what might be called "meta-decision-making" mechanisms. In a number of companies there are cross-functional and perhaps cross-level committees that sort issues and try to develop appropriate processes for each of them. To choose a single example: At a Shell plant in Canada, issues that cannot be dealt with by individual teams of workers go to a "team norm review board" composed of operators, union officials, and managers; this committee then establishes a way of resolving the problem—they may, for example, set up a subcommittee, or a series of meetings of the affected groups, or a call for additional information from inside or outside, or some combination of these tactics.

Principle 8: Though relationships of trust are a critical ingredient in these systems, these are not the warm *gemeinschaft* solidarities of traditional communities, or even of the communal version of bureaucracy. Relationships in such a system are formalize and specialized to a high degree: it is a matter of "knowing who to go to" for a particular problem or issue, rather than a matter of building a stable network of friendship relations. Thus influence relations are wider and more divers, but also shallower and more specific, than those of traditional "community." Mangers in systems moving toward a more interactive form often report a sense of loneliness and isolation, in comparison to the "old days" of communal bureaucracy. Information systems have also facilitated the building

of temporary networks. It is now possible in some companies for mangers to put out a general message asking for help on a given project, or to collect a list of people who have knowledge and experience in the area: they can maintain contacts with people whom they never meet face to face.

Principle 9: In order for a system of influence to function, there must be ways of verifying and publicizing reputations. There must, therefore, be unusually thorough and open processes of association and peer evaluation, so that people get a relatively detailed view of each others' strengths and weaknesses. Perhaps the best example of such systems in industry is in investment banking: in many such firms people work in a wide variety of peer teams and are constantly involved in mutual evaluation.

Principle 10: A post-bureaucratic system is relatively open at the boundaries. A critical manifestation of this is career patterns: Unlike the situation in large bureaucracies, there is no expectation that employees with spend their entire careers in one organization. There is far more tolerance for outsiders coming in and for insiders going to. This ingredient of this pattern has recently caused great distress in the closed communities of the corporate management. The competitive pressures of the 1980s have caused widespread managerial layoffs for the first time, and simultaneously the speed of technical innovation has required increased hiring of outsiders. Both these developments have threatened the "family" atmosphere that has fostered unity and cooperation in the past. A second aspect of "openness" is the growth of alliances and joint ventures among different firms. These have been growing explosively in recent years even among firms, such as AT&T or IBM, which have had a long tradition of "going it alone."

Principle 11: The problem of equity acquires some new wrinkles. In a bureaucratic system, the main touchstone is always objectivity and equality of treatment: Thus, there is a constant effort to devise rules for treatment of employees, and to minimize the element of personal judgment. In a post-bureaucratic order there is first of all an effort to reduce rules, and concomitantly and increased pressure to recognize the variety of individual performances. This is a major point of tension in many innovating companies at the moment. It is likely that he solution involves the development of *public standards* of performance, openly discussed and often negotiated with individual employees, against which they will be measured; this is an apparently increasingly common approach to compensation.

Principle 12: Time is structured in a distinctly different way from a bureaucracy. In the latter, decisions are made with the expectation of permanence: "This is the right way of doing things." Review processes occur at regular intervals, usually annually in a budgeting process, as a way of checking that things are functioning as they should—but not with the anticipation of making fundamental changes. Thus, structural change in bureaucracy comes as something of a surprise, as a dramatic "break" in the flow of events. A post-bureaucratic system, by contrast, builds in an expectation of constant change, and it therefore attaches time frames to its actions. One element in structuring a process is to determine checkpoints for reviewing progress and for making corrections, and establishing a time period for reevaluating the basic direction and principles of the

effort. These time periods are not necessarily keyed to the annual budget cycle: they may be much shorter or much longer, depending on the nature of the task. This flexibility of time is essential to adaptiveness because the perception of a problem depends on putting it in the right time frame. If one focuses—as most mangers do—on the issues that must be dealt with in the next week, on will simply never recognize the existence of issues that have a longer "cycle time"; and of course the reverse is equally true. The ability to mange varying time frames is a major advantage of the post-bureaucratic system.

B. Protocol for Interviews with Committers and Leaders

I. Background information:

What are your age, gender and nationality?

Are you currently a student in a degree or certificate program and in which area?

Are you employed? What is your job title? For what type of organization?

How long have you been employed as a computer programmer or a similar position?

How long have you been a committer/leader in LithiumBSD? What is your current status in the group (committer, core/leader, etc.)?

Are you a member of other open source groups? What is your role in these groups (committer v. just sends in occasional patches)?

II. Open Source Groups as New Work Organizations

What different official or unofficial positions exist within the group?

What technical areas and functions are you responsible for in LithiumBSD? What type of projects do you work on? (Probe for specific examples of day-to-day work) Have you switched between different roles and types of work (developer, manager, documentation, etc.) within the group?

In your experience, how is work divided up within BSD? Who, if anyone, makes sure that important work gets done?

What is the typical way to gain commit and leadership status within Group Name?

How did you come to be accepted as a committer/leader? (probe for specific actions and events that allowed them to gain status in the group, and which individuals determine merit)

How do you recruit new members to Group Name?

Has your participation in this group increased the depth and breadth of your technical skills?

Has it increased your project management skills including leadership and coordinating projects?

How do you coordinate your work with other developers in your functional area (Graphical Interface, Networking, etc.)? With the rest of the team? (Probe for specific examples of how work is delegated and who ensures that it gets done)

Are the types of programming tasks in LithiumBSD that are more desirable to be a contributor for? How does the group decide who gets to control these areas where there are conflicts?

How does the group determine which solutions and priorities are best for the project?

How are technical and personal conflicts resolved? (probe for examples)

Do you think conflicts are typically resolved fairly?

How might you improve conflict resolution?

To what extent do group members work together on specific tasks v. only focusing on their specific programming/documentation, etc. task? What level of peer review is there for changes to the kernel?

How are decisions typically made within the group? (Probe for specific examples) Is there more of a top-down or consensus model of decisionmaking in your group?

Have you ever initiated a new project, process, or other change in the group? (get narrative) Do you feel that your suggestions have been responded to favorably by leaders? By the group? (probe for detail)

How would you improve the process of managing Group Name including the coordination of work and deciding project priorities?

Are you satisfied with current and past leadership in the group? (Along with prior question on conflicts, measures of perceived fairness of processes and organization as a whole)

III. Occupational Communities for Software Development and Project Management

What is your opinion of the move in software toward object-oriented programming (which developers a toolkit of pre-programming tools or “modules,” making

programming more accessible to the layman) and other attempts to standardize and streamline the task of programming?

In your current or future career, would you be willing to take a significant pay cut to work on open source projects rather than on proprietary projects?

Are you a member of any professional organizations related to your work (e.g. ACM)? Why or why not?

What are the main reasons you participate in open source?

Did you get involved in open source to learn new programming skills and styles of coding? Did you get involved to get feedback on your own coding skills?

Why did you choose to participate in Group Name rather than other open source projects? (probe both for ideology of group)

What are the main channels of communication for the group? How much informal interaction is there in the group? Do you tend to communicate with the group as a whole or just a few members? Are private channels of communication often used? How often do you use IRC to communicate with peers?

What factors help to retain good developers within a group and what lead to their departure?

What is the best way to motivate and retain developers within Group Name? Why do you continue to be a member? (Measuring what holds the community together)

What rewards do members gain from gaining a high profile position, such as leadership, within the group?

Would you like to have a more important role or position in the group? If yes, what has prevented you from gaining this position? If no, why not?

What actions, statements, etc. would lead to the expulsion of a member from the group?
What actions, statements, etc. would lead to punishment of a developer?
Has anyone broken these rules? How was it dealt with? (probe for long term result, e.g. reintegration into group after expulsion, and informal punishments such as shaming)

Which rules is it important that everyone obey for Group Name to function well? (probe)

If employed: How would you characterize the management at your organization? Are they technically competent? Can they effectively coordinate and evaluate programmers? How does the process of design, coding documentation and debugging in Group Name differ from industry practices?

Do you think that it's more important for companies to hire programmers based on their reputation among other programmers or based on their past education and credentials?

Both Employed and Not Employed: What would your ideal work situation be? Would you like to become a project manager at some point? Would you prefer to work autonomously with little input from coworkers?

What are the benefits, if any, of the open source model of development in managing software teams?

Are there any important issues related to Group Name that I haven't covered?

C. Brief Statement of Findings from Preliminary Interviews

1. Role and Task Negotiation

Both groups are informally coordinated by small elected bodies, which have little power to enforce decisions. Rather, these leaders attempt to set priorities and develop project for the group, with a minimal level of detailed plans on how they should be implemented. The best way of implementing the project is often negotiated through group discussion or decided by the individual who is perceived to know the most about a given type of problem. Due to the complexities of developing an operating system, which combines a wide variety of functions including networking, graphical interfaces (projecting information onto a monitor), and printing systems, to name a few, it is easy for developers to become overwhelmed by the volume of messages about the project. Specialized mailing lists are an important tool for coordinating work within these specific areas because messages are only sent to interested individuals. According to one respondent, there have been occasional attempts to create a position for an individual who

oversees all changes to the program, but this is generally too time consuming for most members and requires a breath of knowledge not typically available to a single developer.

One major source of conflict between individuals is over decisions regarding whose programming solution will be incorporated into the source code of the program. According to one respondent, minor changes are generally coordinated informally and the person proposing a change will simply email other members of the groups to confirm that his intention will not create problems for the other developers. Major changes, on the other hand, may fuel spirited debate between members who have competing priorities or "philosophies" of development and members discuss these conflicts through email lists in an attempt to convince others that their approach is the best. Whenever possible, members attempt to test competing solutions by developing "benchmarks" which determine which approach is more efficient, secure, etc. and often developers will attempt to implement their plan to prove that their idea works (also know as a "proof of concept").

Although generally there are attempts made to resolve conflicts through consensus and objective standards, resolution of conflicts is not always civil, and occasionally developers will engage in "commit wars" where each individual implements their preferred solution within the development tree, erasing the changes made by the competing developer, or "flame wars" where two or more users attack their opponents within a public forum (typically email lists).

While a certain level of assertiveness is required for a developer to make his opinions heard, developers who pay little heed to the opinions of other members risk various forms of punishment, including temporary expulsion from the group in extreme cases. There is

also evidence of status hierarchies within groups which are evident in the amount of attention that are paid to a members suggestions by the rest of the group. The processes of gaining the initial "commit bid" allowing access to the source code of the project and of becoming and influential member of the group were examined in more detail in the case studies of Star Linux and LithiumBSD.

2. Open Source as an Occupational Community

Based on initial interviews, an important motivation for volunteering in open source groups appears to be gaining a level of autonomy and opportunity to work with a community of their peers which are not available within the typical work setting or educational program. It appears that many programmers participate in open source groups to gain the camaraderie and exposure to new ideas that are only possible within a community of one's occupational peers, as well as exposure to interesting programming problems to overcome the monotony of work or school. As one respondent explained, "if you're sitting alone in your office and doing your job, maybe your boss is really happy with you and is really happy with the work that you're doing but he doesn't care about the code that you're writing." Generally, most professional settings do not provide a mechanism for critical feedback regarding programming skills, and looking at the code of other programmers in the organization may, in extreme cases, be actively discouraged and schools typically do not actively encourage peer review of coding styles. This motivation for participating in open source is explained by the same respondent:

> We don't care what the marketing people say. We don't care if the man on the street doesn't like what were doing. We're not writing code for that

> guy, we're writing it because of some problem we want to solve. We have this job we want to do, and if the code does our job in a way that we like, then we don't care if someone who doesn't need that software does not like how we did it.
>
> But at the same time, it's certainly true that part of the motivation for participating "as a member of the project" is that participation is a way to get praise from people whose opinions you do care about, which would be other experienced programmers.

The case studies of LithiumBSD and Star Linux further examine whether open source functions as an “occupational community,” for example by providing peer review, skill development and mentoring to members.

C. Midpoint Analytical Note

1. Reasons for Participating in Open Source Projects

The open source movement seems to serve several different functions for participants. For many it appears to be an escape from traditional management models which allow decisions to be made based on profit, or social/political reasons rather than the ‘best solution’. However, these projects do not seem to form a real ‘social movement’ in response to the IT industry for most people. Many of the participants in open source are happy in their current jobs and are able to use open source tools in their workplace. One respondent who works as a senior programmer at a local university is able to work on open source projects in the course of his other duties, because he is able to use his skills and the programs that he creates to solve university problems.

For most programmers open source development is both a hobby or an intellectual pursuit that they find fascinating, as well as a way to increase both the breadth and depth

of their professional skills in a way impossible in both school or in work roles. Participants seem to choose their projects because they find them interesting and desire how to 'program the right way' and to learn from their peers. Another main motivation is that open source programming seems to give programmers access to a wider range of roles, tasks and projects than they would have in school or work and this can increase their programming skills.

Several developers believe that open source programming teaches them different styles of coding and different ways of solving programming problems. For example, while a particular problem can be solved using several different programming languages, each solution may be better or worse depending on the priorities of develops. Developers then seek experience with a wide range of programming languages (C++, Python . . .), 'styles of coding', platforms (PC, Mac, Linux . . .), etc. even if they have no immediate use for the knowledge. They explain this wide-ranging interest either as intellectual curiosity or as a way to increase their 'flexibility' or 'range' in programming.

Developers with competing solutions for projects often learn a great deal about programming when they examine other solutions to a problem that they were working on. Developers working within a corporate environment will not typically have this wide ranging experience, because they typically focus on one project or type of programming. In school, sharing of code and competing solutions also does not typically occur, in part because instructors tend to teach a single approach towards coding and because reading others code can be seen as cheating.

Generally motivation for participating in projects is not 'utilitarian' (increasing one's skills to get a better job) but nearly all interviewees cited this as a nice additional benefit.

Membership in a high status group can also increase a programmer's status among her peers and within the professional community, especially if she is recognized as a major contributor. (See Hierarchy and Conflict section)

Other motivations for participating in these projects include socializing with individuals who share an interest in the technical world and being part of a professional community and some participate for more ideological reasons such as "making information free" for the masses or undermining closed source companies such as Microsoft.

2. Project Management

One major reason that individuals participate in open source groups is to gain freedom from traditional management structures, where decisions are not made based purely on technical merit. As several interviewees pointed out, open source is not free of these problems. However, leaving problem groups is much easier in the open source world than in the IT industry. This has led some observers to argue that the Open Source community, which includes many thousands of projects, is evolutionary in nature, because only the projects with the best management structure retain the most proficient programmers. One interviewee explains that he prefers to work in an environment with little or no management interference:

> My immediate boss isn't a programmer. He's got somewhat of a technical clue but he's not a real programmer or anything. I like the fact that he's very hands off. It can be a good or bad thing, I like it but I know that some previous employees have criticized him because he's so hands off to the point where you don't have do anything if you don't want to. But for me,

> I'm very self-motivated so I enjoy it and I'm able to just take initiative and just do things that probably need to be done.
>
> And he handles all of the administrative overhead. He handles making sure people get paid, taking care of the paperwork, making sure that his bosses are happy and making sure that there a people who are actually going to use the work that we are doing. We're able to focus on the technical problem and he handles the managerial stuff.

And another interviewee contrasted the project management style of software companies with open source groups, and he explained that the corporations were typically less organized than the open source groups:

> They're typically small companies that I work for and their main focus wasn't really software development so the manager-type people had no clue about software development and they dictate how you work, more or less. In open source there's no management, you do what you think is best, or you do what the project leader thinks is best, but the project leader is typically the head developer.

Open source groups typically have a meritocratic system of leadership where an individual's standing in the group is based on their ability to write usable code and to solve difficult programming problems. Programmers who are fed up with the managerial politics of their current job, or who want to test their skills in difficult projects, find this lack of managerial structure refreshing. As one participant explains:

> As a matter of fact, a lot of open source people work in [the IT] industry. It's a way that they can go to work and instead of taking orders from people who don't know what they're doing they can say "this is the way things work, I have the experience." In addition to doing the work, I know how the work should be done.

Developers often choose between competing projects based on the level of appreciation that they receive for their efforts. Leaders who recognize and reward technical merit tend to retain the best developers and these tend to be the managers with

the best understanding of the technical side of the business. In response to the question ‘what makes a good project leader?’ one interviewee explained:

The most clueful person makes the best project leader. Whoever knows the most makes the best project leader, that's all there is to it because they're the one that's going to be doing most of the code. It's not a project leader as in telling other people what to do or giving out tasks, because that's not the way it works. The project leader is the person who decides which direction the project is going in and does the most code, so the smartest person obviously.

3. Work Roles and Coordination

For many users, participation in open source groups give them a wider range of technical and management skills that they would not gain in either work or school. There is usually not a fixed hierarchy within projects and a limited number of formal rules and this allows participants a good deal of freedom in negotiating tasks and roles within an existing group, especially if they have are skilled programmers[82]. Developers can contribute to a project whenever they wish, and they decide which tasks they work on, and when the task is done. Unlike traditional jobs, members can easily leave a group if they dislike the decisions that leaders are making, or the rules that they set, and they always have the option to start their own project. Since projects require a large pool of developers, it is likely that the most popular projects are those with the most transparent management, or ‘enabling bureaucracy’ as defined by Adler and Borys. These

researchers argue that enabling bureaucracies tend to create rules and to use technologies in ways that "help users form a mental model of the system they are using"[83] by making the underlying structure of the technology clear to the user. Similarly, rules and roles, which are clearly related to organizational objectives, will be more 'enabling' to workers and less 'alienating'.

Coordination problems are the major disadvantage of the organizational freedom of open source. In traditional companies, it is possible for a manager to delegate work roles, tasks and deadlines to ensure that there are no wasted efforts or missed problems. When developers in a group are volunteers and are scattered around the world, it is much more difficult to manage them using a traditional model. As one interviewee pointed out "no one is open source has a big enough stick" to force developers to complete certain tasks.

As a result, two major problems arise. First, jobs that are not fun or intellectually challenging (such as creating documentation for users) typically receive the least effort and second, developers with similar interests may run into conflicts when they disagree on the direction a project should take. In a traditional business, the first problem would be solved by delegation and the second by managers mediating the problem or threatening punishment. In open source this lack of coordination and hierarchy can lead to problems such as 'commit wars' where a developer will commit his code to the source tree, only to find a short time later that his changes were overwritten by his rival. In the LithiumBSD project, one attempted solution to this problem has been to assign one or two committers to different areas of the kernel (such as networking) so that developers who want to make

[82] As the size of the project increases, elaborate roles and rules are often developed. For example, Debian, a distribution of Linux has an explicit constitution, mission, and well defined rules for becoming a member, and well defined processes to ensure that developers efforts are well coordinated.

changes to that system will need to seek their approval first. The same group also has a fairly complex system of organization where only 300 committers can make changes to the program, and they are coordinated by 'core', a group of seven developers who have a broader view of changes to the system, and who resolve conflicts between developers.

One interviewee explains that this system of organization is needed because:

> There are only so many people you want actually want working on the base source because everybody starts tripping over everybody else, so you have to have some group that you can trust to work together and stay out of each other's way as much as possible.
>
> That group for LithiumBSD is about 300 people and those 300 people are spread literally all across the world. LithiumBSD also has a leadership body, which is sort of the board of directors for the whole project, and there are nine people, nine positions in LithiumBSD core. The way that you get into LithiumBSD leadership is the other 300 developers elect you. The people who are in LithiumBSD leadership right now include people from Australia, California, New York City, England, and some former members are in Denmark, I think one of the former members is in the Netherlands. We have some people who contribute who are from Russia, so it really is available to anyone who wants to do the work to contribute.

Smaller groups are much less concerned with models of formal organization, in par because they often are focused on simpler programs. One interviewee explains how she organizes the small group that she leads:

> Every person does what they want to do. Let's say that we have an operating system and it lacks a feature, or there's something that a specific person wants. That specific person writes the code because it's something they want. That's the only way that good code gets written.
>
> There's no, "everyone sits down, you do this, you do this." It's "I want this, so I'm going to write it, that's it" It's the only way that's really accepted in the project that I'm working on. Otherwise if you don't want to do, you're not going to do it well, so give it t someone who wants to do it.

[83] Paul Adler and Bryan Borys, "Two Types of Bureaucracy: Enabling and Coercive," ASQ, 41 (1996): 61-89; Quote cited, pg. 70.

In my two case studies, I will provide more detail on formal organization and roles. At this point I'm planning to focus on two groups developing operating systems, which tend to require the most coordination because they deal with many aspects of the computer system including networking, memory management, and graphical interface for the user.

I. Contrast with Roles and Tasks Taught in School

I didn't get a great deal of information about how coding is taught at school, but several student involved in open source responded that it was too 'theoretical' and that grading is typically based on following the models taught in class, even if they don't work well in practice. However, programming classes do teach a disciplined model of design, which might be lacking in coders primarily learn how to program 'on the fly' through open source projects.

In addition, students rarely have an opportunity to work with or get feedback from other programmers through classes, in part because it is difficult to grade their contributions to the project and because of concerns that students might 'cheat'. As a result, students get little critical feedback on their programming, except from the professor who often will only tell them "what they did wrong." Several interviewees complained about not learning about the many viable solutions to a programming problem that are ignored in favor of the professor's preferred approach.

Several students complained that their classes were too theoretical with one interviewee explaining that "I honestly don't care about the theory that's not useful to

what I'm doing. I like to actually create things that are useful." Another student argues that the model of development taught in school is inferior because:

> At school it's mostly theoretical, it's not actually release-style code. It's not something that you could actually give to something to actually use, because there's no error checking. You expect things to work based on your theory and if they don't work, oh well. You throw out all the other cases. When I'm doing open source stuff, it has to work all the time. There's no room for it to break, because then people won't use it at all.

These practical skills will benefit students with open source experience when they enter the IT industry.

In open source, practical solutions to problems are more valued than abstract models. To decide between competing solutions to a programming problems, open source developers must provide a 'proof of concept' which shows that the their idea works in practice and whenever possible 'benchmarks' are developed which objectively measure the performance of the solution. This is in stark contrast to both 'theoretical' coding in classes where there is an emphasis in coding according to the theory taught in class (even if it doesn't work in practice) and on 'doing it the right way'. Also, this contrasts with models of coding used in work, where decisions about the 'best solution' are typically made based on the customers needs combined with managerial decisionmaking which may include factors such as finishing within a deadline, or political or social considerations rather than 'the best solution'. (See next section)

Finally, several students argued that while classes excel at teaching "mechanical" tasks, such as using the right syntax while programming (organizing the program so that it follows a well accepted format), they cannot teach the more 'organic' or intangible skills such as debugging (See Burns and Stalker 1966), which require either an

understanding of the system as a whole or practical experience with similar problems. As one student explains “Design is a creative process and is not taught well at ----" These skills are more easily learned through participating in a wide variety of projects and roles, a level of experience that would otherwise not be available without years of experience working in many IT positions.

II. Contrast with Programming at Work

Participation in open source projects give developers a wide range of programming skills and allows the freedom to work within many different roles in a project. I found that many of the open source developers that I interviewed had varied roles across projects, from leading their own smaller projects on one hand (which required an understanding of the project of the whole), to working on a specific task within a larger project.

Workers within traditional businesses tend to focus on one project or type of programming, or to have a specific function (e.g. maintaining the servers and network) and this can ‘limit their horizons’, as one interviewee pointed out. This interviewee, who has been highly involved in maintaining operating systems and networking at a local university provided the following anecdote to explain the importance of having a wide range of experience as a programmer:

> When I was young, and had just been hired at my first job here at ----, there was someone who came in for a job opening that we had. It was a job position for doing low-level systems programming, which was mostly in IBM/370 assembler. This guy came in who had just been fired. He was in his mid-fifties, and had worked 20 or 30 years at some company. In all that time, he had worked on one specific project, probably something like

> a custom payroll system written entirely in Cobol. He had been paid quite well, and had had no problems with the company for all the time he worked there. But then they bought some new payroll system off the rack, and didn't need him to work on that custom program anymore. And with that project gone, there was nothing in the company that he had any experience for, so they fired the guy. They gave him very nice reference letters, but they simply had no job for him anymore.
>
> So, he comes to apply for this job with us. He has no idea of systems programming, he has never programmed in assembler, he didn't work on IBM computers at all, and he didn't know *any* of the tools we used for our development. So for me (as a hot-shot kid out of college), it was pretty easy to see this as a funny situation. In the interview it was clear that he was completely unqualified for the job -- there was no way he could even understand what we were doing. But on the other hand, my own father was about the same age as this guy was at the time, and I couldn't help thinking that if it had been my father, the situation certainly would *not* have been funny.
>
> For him, he's out of work and has no marketable skill. If he had asked me if I knew anyone who would hire him, I could not have even guessed at anyone who would hire him for any job in computing. When you're fifty and suddenly out-of-work, it isn't funny. So that's part of the reason I work with open-source projects. For one, it forces me to keep my horizon broader, because I have to pay attention to what people are working on outside of my own company. If no one in the open-source world cares about I'm working on, then I better start working on the projects people do care about. For two, I do actually get job offers based on nothing more than source code which I have written and made publicly available. I am sitting here in ----, I have never written a resume in my life, and I've gotten at least 20 job offers based on somebody seeing my source code. And most of those job offers would have paid me more than I get at ----. So, in that sense, I could have made more money by being associated with open source, but I stayed working at ----.

The interviewee explained that since he is perceived as valuable (due to the wide range of his programming skills), he is more able to say “this is what I believe, this is why I believe it, and [I’m] not afraid to bring that forward.” This respondent believes that he made the right move by staying in his current position because the other job offers

gave him more autonomy in his current position because they realize the value of his skills.

4. Hierarchies of Knowledge and Group Conflicts

Every interviewee with any practical experience in Open Source groups pointed out that projects were not motivated simply by altruism and that group and personal conflicts were fairly common. As a few people pointed out, this is true of almost any group of people working together, which contrasts with the ideal that participants will gain status in groups and in the community only based on the quality of their code. Generally, the groups which retain developers and to last over the years do seem to function as meritocracies, or at least as 'benevolent dictatorships'. A committer in the LithiumBSD group explains the process of gaining status in the group in this way:

So, then the question is how does one go from being "some unknown guy" to being someone who would be recommended as a committer.

The most popular way is to send enough problem-reports of high enough quality that other committers start to notice that you have the ability to really contribute to LithiumBSD. Not just that "you would like to," but that you can write code on your own which would improve LithiumBSD. They also notice how you behave on the mailing lists, and decide "is this a guy that we would like to work with?." That describes how it works for source-committers. There are different rules for doc[umentation] and ports committers. Different groups of people [not just "core"] will select who can be added as committers to those categories of work.

This was the process that he followed to receive 'commit status' or the ability to modify source code himself. No one else was really interested in the type of problems that he was interested in (printing systems) and after submitting multiple patches, one of the other committers offered to make him a committer. Membership in high profile projects gives programmers status, which may lead to later job offers.

Interviewees with practical experience repeatedly explained that few developers are motivated by altruism and that personal conflicts in groups are far from uncommon, especially in larger groups (only those with little experience in open source groups seemed to believe otherwise). Conflicts occur based either on technical disagreements over which programming solution is the best, or personal disagreements. The former type of conflict is settled through 'benchmarks' or logical proof whenever possible (objective measures of performance once the code has been written).

Personal conflicts are more difficult to deal with, especially since the most headstrong developers also tend to be the most skilled. One interviewee described these individuals as 'code bullies' who are not responsive to the suggestions of others. Another developer describe how his 'bug fixes' for two separate projects were ignored by leaders, who proceeded to make a similar change without giving him credit for pointing out the problem. In the long run personal conflicts can destroy a group, leading to a 'fork' where the group splits into two competing camps, as illustrated in the quote below:

> The biggest thing you wind up getting is squabbles over people's code and their relative merits, unfortunately. It's the reason why LithiumBSD exists today . . . But there were problems within the community. William is very headstrong. William left and started up his own little Wonderland project [LithiumBSD]. So you'll get things like that where someone doesn't think they're contributing as much as they are or their ideas aren't being heard and they felt they are very extremely valid ideas, you have a potential for a schism. And that's where you have the potential of losing a large group of

> developers. You'll have a large group of developers, who because they're good are extraordinarily headstrong, they want to push forward their agendas and then you have the people who aren't like that. And the people that you wind up losing are the headstrong people because they felt "this is something that I really wanted to be a part of but no one cares what I think anymore," and they all go their separate way, to generalize a great deal.

Another interviewee explains that groups that don't make decisions based on the 'best solution' cannot survive in the long run:

> In the case of LithiumBSD literally was one person all the time [who was causing group conflict], it was William. In other cases you might have a whole bunch of people and it is whoever shouts the loudest and the winner will he different depending on who feels the strongest and that certainly is not the same as staying the winner is the one with the best technical solution. It's just the one who has the most vicious feeling in his gut that he's the right guy and the other one is wrong, based maybe on nothing other than he hates the other guy on a personal level. But if he's the one that shouts the loudest, he's the one that wins. And those are the projects that if that isn't corrected eventually they blow themselves apart, one way or other because as human beings you just can't win in a fight all the time, even if you're winning some of them.

The main source of conflict is often the most skilled developers, as this interviewee points out:

> In the last year or so of LithiumBSD there's been an increasing amount of this friction between developers. Of particular importance is that it's between developers who are doing a lot of work that everyone would like to see done, so you really don't want to simply kick the developer out of the group. At the same time, they infuriating several other developers who may be "lesser" developers, but some of them are "lesser" developers only because they're new to the project. Maybe a year from now they would be a great asset to the project, but they will get so fed up that they won't stick around that long. They'll just say "enough with you guys, I'm not going to spend all my free time fighting, so I'm out of here." And they'll go off to be a great guy for somebody else's project.[84]

The projects which are able to negotiate these conflicts of roles, tasks and

[84] The prior quote is also from the same individual

deciding who has the 'best' solution tend to survive the longest[85]. It is also possible to see these conflicts as a process of renegotiating work roles in an environment that lacks a formal hierarchy, similar to Calvin Morrill's study on "Conflict Management, Honor, and Organizational Change"[86]. This interpersonal experience is valuable for programmers developing their professional skills, and negotiating their career within a somewhat fledgling industry where work roles are not always clear[87].

5. Open Source as a Foundry for Professional Skills

Both students of computer science and workers in the IT industry explained that open source projects and groups gave them hands on experience solving software problems, and coordinating work with others. When I asked one student what he gets out of the groups and projects that he participates in, he cited the managerial and practical skills that he gained that will come in handy in his later career:

> I definitely took to the ACM because I liked being surrounded by competent people. I liked feeling like I was really learning something that was useful. I'm not convinced that ---- does a very good job of teaching a lot of the time. I definitely noticed that everybody around me in the ACM, definitely freshman year, everybody was smarter than I was and everybody used so much more and wanted to learn as much as possible about computers. So I stuck with the group and picked up a lot. By the end of freshman year I had picked up so much it was amazing.
>
> And afterwards I ended up taking over the ACM chairmanship after that year. I couldn't believe that I was being put in that position. But I did the best with it and I learned a lot about leading a group and about trying to . . .I remember that D. R. and I got in some huge fights that year. I hadn't

[85] The solution to conflict may be far from democratic, for example OpenBSD and Linux are often considered "benevolent dictatorships" or "cults of personality" where a single leader has a large role in decisionmaking.
[86] American Journal of Sociology 97 (1991):585-621
[87] See Barley, 1996: 404-441.

> really had to work closely with others before and had no experience in delegating responsibilities, I made so many mistakes, and I learned a lot. I think that at this point I've fallen in love with a group I enjoy. I really enjoy working with the folks and I like giving back.

It is unlikely that he would gain these interpersonal skills as easily though school or work. The following quote from the same individual shows that these conflicts can have positive benefits in learning interpersonal skills. The same interviewee continues:

> I know in the ACM there were many conflicts due to whose job was whose, and kind of boundaries and borders between jobs. I know when ---- and I fought a lot with the ACM stuff, we just eventually learned how to work with each other and pull back a little bit and just not mess with each other as much. But I haven't had too many conflicts that I've had to worry about, at least with the technical projects that I'm working on. But most of my projects have been very small in scope and very small groups so it hasn't been a problem. There is definitely some give and take that has to be learned and if people aren't willing to compromise, the projects can be forked and this could be good or bad for the group or the project and there are counter examples of technical projects that have been forked because of ego and ability to work with others and irrelevant ideologies.[88]

And another observer argues that to gain recognition, technicians need to learn how to voice their opinion and become more assertive in their roles within business and the economy:

> There are a lot of blatant hot heads in the open source community. It's actually probably one of our biggest issues. There are people who think that they are gods, and that's it. You get that in any community. I wouldn't say that it's a function of the open source movement, although they're probably more vocal because they're a group that's trying to gain recognition and acceptance and right now they have a large segment of the economy going against them. So they have to, to a certain extent be hot

[88] Smaller groups tend to be less hierarchical and to have fewer conflicts. For the groups that were discussed in the interviews, this lack of conflict was explained by the fact that members already knew each other and that they all agreed on the direction that the project would take. A leader of a small project explains: The thing is that none of us are working on any project where the developer lists extend outside of ourselves. It's more like we'll go to other people to suggest things, but we're the primary developer. If someone can suggest something reasonable, we go with it or not, but we don't have arguments about things. There aren't two developers who are working head to head on something.

> headed . . .but it’s just a function of humanity as opposed to a particular feature of the open source community I would say.

The process of negotiating these roles within open source groups, and to a lesser degree within the IT industry, will be studied in more depth in the case studies of LithiumBSD and Star Linux.

Appendix D: Example Work Practices Described by Members

A typical member of LithiumBSD provides a summary of his daily work activities within the group and explains that his typical process of work requires a high level of communication with other developers:

> There is a mailing list called "lithiumbsd-xtz" on one of the LithiumBSD machines. Technically it's a public list that anyone can join and post to, but in practice it's just me and him talking. So, I start work on some update, post a message, he reacts to it, I make changes, and eventually we get to a point that we're both pretty happy with it. I commit my change to xyz in the "current" branch of LithiumBSD. "current" is explicitly the cutting-edge branch, and it gives a little bit of a testing ground for a change. When a developer does a commit, there is a template of information, which pops up with various fields the developer can choose to fill in. One of them is "Reviewed by". I say "Reviewed by: pingman", and there's pretty much no one who will complain. Between me and him, there isn't anyone in LithiumBSD-land who knows anything about xyz that we don't know. So, in some sense that's hard, and in some sense it's easy. It's hard because Garrett always has good ideas, which I feel obliged to include, so the review process takes awhile. Maybe a week or two. But once I get past him, it's easy sailing. For other things I change, I have to search around for someone who might care. This process also has a bunch of waiting in it. "Post an update, and wait." "Gather feedback, change the update, repost, and wait." "Commit to current, and wait." These waiting periods mean that you often want to start on something else, and then you find you're juggling three or four completely unrelated changes all at the same time.

These work processes appear to be typical of the processes used by members of other large open source groups. One member of Star Linux and several high-profile groups details his overall work on the project and then his daily tasks for each project:

> I do a number of things: 1. Bug fixing as appropriate 2. Suggesting improvements that would allow us to implement a new feature or maintain something more easily. 3. Suggesting new architectural directions. . . .I also do a few odd jobs around the edges, and try to write a bunch of useful documentation.
>
> So, actual daily processes of what I do is: 1. Pull in the latest CVS tree which involves synchronizing my copy of the source tree(s), there are three for Samba, one for Ethereal, with the one kept up on the CVS repositories. This makes sure that I have all the changes committed by everyone else. 2. Make any bug fixes, either in the code or the infrastructure 3. Make sure it builds, which entails running the build script, and if it fails, making sure I pulled in everything. This might involve re-running the CVS command to create directories I missed. However, it might also involve fixing assumptions made by the committer that are wrong on my platform. 4. Submit the changes back. On average this will happen about once a day, but it is bursty. Some weeks I might only make one change for that week. Other times it might be several a day. In addition I work on maintaining communications, especially in the Samba team. This takes the form of team email, email to the samba-technical mailing list, and discussion on the #samba-technical IRC channel.

Appendix E. Glossary

ACM: Association of Computing Engineering. A professional association for computer programmers. Organizes and funds open source, and other computing groups on many campuses.

BSD: Berkeley Software Distribution. An open source operating system that was a precursor to Linux. Distributions include NetBSD, FreeBSD, and OpenBSD.

Coding: Same as programming.

Committer: An individual with ‘commit status’ within a group. Committers can make unrestricted changes to the source code of a program. Individuals without this status must submit patches to be included in the source code, at the discretion of a committer.

Developer: Same as programmer.

Distributions: Modifications of the source code to operating systems, etc. made by different groups who often have different priorities or styles of coding (e.g. security versus efficiency versus ability to work on many different types of systems)

Documentation: Writing documents for either other developers, which explains the logic behind a program (e.g. comments in the source code), or for users of a program (e.g. user manuals or FAQs).

Email List/ Listserv: A mailing list that allows any member of the list to distribute comments to other members. Often geared towards specific interests (e.g. creating a version of Linux that work well for schools).

Flamewar: The exchange of insulting or critical comments by email or on an email list between two or more individuals.

Fork: When an open source group splits into two separate groups due to personal or technical conflicts.

GUI: Graphical User Interface (pronounced 'gooey'). Allowed for the birth of the modern computer by allowing users to operate programs using graphical displays on the monitor (e.g. pull down menus and windows) rather than a text-based, 'console' interface. Especially important for programs that want to be 'user friendly'; often hackers prefer to use programs which lack GUIs but have more functionality.

Hacker: An individual with at high level of technical skill who is interested in understanding the 'guts' of technical systems and building their own 'hacks' to existing systems. Hackers generally have a disdain for 'closed technology' (e.g. proprietary software) that they can't examine or modify. The popular meaning of 'hacker' as a malicious individual intent on breaking into computer systems and writing viruses and other exploits is a more recent use of the term. The open source community refers to this group as 'crackers' and generally distances themselves from them, although there is some overlap between the two communities.

IT: Information Technology. Generally used when referring to the sector of the computing and other industries focused on software development and maintenance rather than hardware, networking or peripherals. One of the fastest growing sectors in the world economy.

Jargon File: An extensive glossary of hacker-related terminology. Online at: http://catb.org/~esr/jargon/html/go01.html See also: http://catb.org/~esr/jargon/html/

Kernel: The core of an operating system that contains the basic instructions that tell a program what to do—the source code. Packages such as word processors and window managers are added to allow users to perform different tasks.

Linux: An open source operating system created by Linus Torvalds and a host of other open source developers. Earlier open source programs provided several critical functions for Linux including Richard Stallman's GNU kernel (See http://www.gnu.org/gnu/thegnuproject.html)

Open Source: Software that allows any individual to access the source code to the program and modify it to suit her needs.

Package: A program that is added to the kernel to increase to functionality of an operating system (e.g. word processing or database programs). One of the main differences between different distributions of the same operating system is in which programs are included with the kernel.

Patch: A modification to a program, either to add a new function or to fix a bug. Submitted to committers, by individuals outside of a group, for inclusion within an open source program.

Release: A new version of an open source program, which is 'released' to the general public when either the group as a whole or the leader(s) of a project agree that it is safe to do so.

Source Code: The instructions that tell a program what to do. This code is modified by committers who are authorized to change the software.

Bibliography

Abbott, Andrew (1988) *The System of Professions: An Essay on the Division of Expert Labor*. Chicago University Press.

Adler, Paul S. (2001) "Market, Hierarchy, and Trust: The Knowledge Economy and the Future of Capitalism" *Organization Science*. 12, 2: 215-234.

Adler, Paul S. and Borys, Bryan (1996) "Two Types of Bureaucracy: Enabling and Coercive" *Administrative Science Quarterly*. 41, 61-89.

Albrecht, Allan J. and Gaffney, John E. (1983) "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation" *IEEE Transactions on Software Engineering*. SE-9, 6. November.

Atkinson, Paul (1983) "The Reproduction of the Professional Community" in *The Sociology of the Professions*, Dingwall, Robert and Lewis, Philip, eds. Pgs. 224-241. Macmillan: London.

Attewell, P. (1987) “The Deskilling Controversy,” *Work and Occupations*, 14: 323-346.

Barker, James R. (1993) "Tightening the Iron Cage: Concertive Control in Self-Managing Teams" *Administrative Science Quarterly*. 38: 408-37.

Barley, Stephen R. (1990) “The Alignment of Technology and Structure through Roles and Network” *Administrative Science Quarterly*, 35, March (1990), 61-103.

Barley, Stephen R. (1996a) *The New World of Work*. British-North American Committee: London, UK.

Barley, Stephen R. (1996b) "Technicians in the Workplace: Ethnographic Evidence for Bringing Work into Organization Studies" *Administrative Science Quarterly*. 41: 404-41.

Barley, Stephen R. (1990) “The Alignment of Technology and Structure Through Roles and Network” *Administrative Science Quarterly.* 35: 61-103.

Barley, Stephen R. and Kunda, Gideon (1992) "Design and devotion: The ebb and flow of rational and normative ideologies of control in managerial discourse." *Administrative Science Quarterly*, 37:1-30.

S. R. Barley and Julian Orr. (1997) "The neglected workforce: An introduction." Pp. 1-19 in S. Barley and J. Orr (ed.) *Between Craft and Science: Technical Work in U.S. Settings*. Ithaca, NY: ILR Press.

Barley, Stephen R. and Tolbert, Pamel (1991) “At the Intersection of Organizations and Occupations,” *Research in the Sociology of Organizations*, 8:1-13.

Becker, Howard (1956) “The Elements of Identification Within an Occupation” *American Sociological Review*. 21: 341-48.

Becker, Howard and Geer, Blance (1970) "Participant Observation and Interviewing: A Comparison” in *Qualitative Methodology*, William Filstead, ed. Markham: Chicago.

Bell, Daniel (1999) [1973] *The Coming of Post-Industrial Society.* Basic Books: New York.

Benkler, Yochai (2001) "Coase's Penguin, or Linux and the Nature of the Firm." Presented at Conference on the Public Domain, Duke Law School, November 9th.

Blau, Peter M. and Schoenherr, Richard (1971) *The Structure of Organizations*, Basic Books: New York.

Bleecker, Samuel E. (1994) "The Virtual Organization" *The Futurist*. March-April: 9-14.

Boehm, Barry (2002) "Software Engineering is a Value-Based Contact Sport", *IEEE Software*, 19-5, Sep/Oct 2002.

Boreham, Paul (1992) "The Myth of Post-Fordist Management: Work Organization and Employee Discretion in Seven Countries" *Employee Relations*, 14 (2):13-24.

Bourdieu, Pierre (1996) *The Rules of Art: Genesis and Structure of the Literary Field*, Trans. by Susan Emanuel. Stanford UP: California.

Brain, David (1991) "Practical Knowledge and Occupational Control: The Professionalization of Architecture in the United States," *Sociological Forum*, 6(2): 239-268.

Braverman, Harry (1998) [1974] *Labor and Monopoly Capital: The Degradation of Work in the Twentieth Century*. Monthly Review Press: New York.

Brooks, Frederic P., Jr. (1995) [1975] *The Mythical Man Month: Essays on Software Engineering.* Addison-Wesley Pub. Co.: Reading, Mass.

Brooks, Frederick (1987) "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer Magazine*, April 1987: University of North Carolina at Chapel Hill.

Bryant, Randal E. and Vardi, Moshe Y. (2002) "2000-2001 Taulbee Survey: Hope for More Balance in Supply and Demand" *Computing Research News.* March, pp 4-11. Also online at: http://www.cra.org/CRN/articles/march02/bryant.vardi.html

Bucher, Rue and Strauss, Anselm (1961) "Professions in Process" *American Journal of Sociology*, 66-4: 325-34.

Burris, Beverly H. (1993) *Technocracy at Work*. State University of New York Press: Albany.

Businessweek (2003) "Programmers are Like Artists: Special Report, The Linux Uprising", March, 3, 2003. Online at: http://www.businessweek.com/magazine/content/03_09/b3822619_tc102.htm

Castells, Manuel (1997) *The Rise of the Network Society.* Blackwell: Malden, MA.

Collins, Jim (2001) *Good to Great: Why Some Companies Make the Leap . . .and Others Don't*, HarperCollins: New York.

Computers and Security (2001) "White House Supporting Open Source Code" v.19, no. 7, News: 577-78.

Conn, Richard (2002) “Developing Software Engineers at the C-130J Software Factory.” *IEEE Software* 19(5): 25-29.

Corbin, Juliet and Strauss, Anslem (1990) “Grounded Theory Research: Procedures, Canons, and Evaluative Criteria” *Qualitative Sociology* 13, 1: 3-21.

Curtis, Bill, et. al. (1988) "A Field Study of the Software Design Process for Large Systems" *Communications of the ACM.* 31, 11: 1269-1287.

Cusumano, Michael A. (1998) *Microsoft Secrets: How the World’s Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*. Simon and Schuster: New York.

Davies, Celia (1983) “Professionals in Bureaucracies: the Conflict Thesis Revisited” in *The Sociology of the Professions*, Dingwall, Robert and Lewis, Philip, Eds. Pgs. 177-94. Macmillan: London.

Denison, D. R. (1990) *Corporate Culture and Organizational Effectiveness,* Wiley: New York.

DeSanctis, Gerardine and Monge, Peter (1999) “Introduction to the Special Issue: Communication Processes for Virtual Organizations” *Organization Science.* 10, 6: 693-703.

Drucker, Peter (1998) *"*Management's New Paradigms*"*, *Forbes*, Oct. 5, 1998

Easterbrook, Steve (1994) Resolving Requirements Conflicts with Computer-Supported Negotiation. In M. Jirotka & J. Goguen (eds) *Requirements Engineering: Social and Technical Issues*, Pps. 41-65. London: Academic Press.

Ensmenger, Nathan "Software as Labor Process" in *Mapping the History of Computing: Software Issues*. " U. Hashagen, R. Keil-Slawik, A. Norberg, eds. Springer-Verlag: New York, forthcoming.

Ensmenger, Nathan (2001) "The Question of Professionalism in the Computing Fields," *IEEE Annals of the History of Computing*. Institute of Electrical and Electronics Engineers: New York.

Faraj, Samer and Sproull, Lee (2000) "Coordinating Expertise in Software Development Teams" *Management Science*. 46, 12: 1554-68.

Fine, Gary Alan (1984) “Negotiated Orders and Organizational Cultures” in the *Annual Review of Sociology*. v.10:239-62.

Fordhal, Matthew (2002) “Rivals to UNIX Servers Gaining.” *Albany Times Union* 15 Oct. 2002: Business.

Freidson, Eliot, ed. (1963) *The Hospital in Modern Society.* The Free Press of Glencoe: London.

Freidson, Eliot, ed. (1973) *The Professions and Their Prospects.* Sage Publications: Beverly Hills: California.

Galegher, Jolene, et. al. (1998) "Legitimacy, Authority, and Community in Electronic Support Groups," *Written Communications*, 15(4): 493-530.

Gibbs, W. Wayt (1994) "Software's Chronic Crisis" *Scientific American*. September, 86-95.

Glaser, Barney G. and Strauss, Anselm L. (1999) [1967] *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Aldine de Gruyter: New York.

Glass, Robert (1998) *Software Runaways*. Prentice-Hall, Inc.: New York.

Gordon, G. and DiTomaso, N. (1992) "Predicting Corporate Performance from Organizational Culture," *Journal of Management Studies,* 29:783-98.

Goth, Greg (2001) "The Open Market Woos Open Source" *IEEE Software*. March/April 2001. 104-107

Gottfried, Heidi (1995) "Developing Neo-Fordism: A Comparative Perspective" *Critical Sociology*. 21-3:39-70.

Gouldner, Alvin G. (1957) "Cosmopolitans and Locals: Toward An Analysis of Latent Social Roles," *Administrative Science Quarterly* 2: 1-29.

Greenbaum, Joan M. (1979) *In the Name of Efficiency: Management Theory and Shopfloor Practice in Data-Processing Work*. Temple UP: Philadelphia.

Greenwood, Ernest (1957) "Attributes of a Profession," *Social Work,* 2: 45-55.

Hall, Richard H. (1968) "Professionalization and Bureaucratization," *American Sociological Review* 33: 92-104.

Hall, Richard H. (1969) *Occupation and Social Structure*. Prentice-Hall: Englewood Cliffs, NJ.

Hall, Richard H. (1991) *Organizations: Structures, Processes and Outcomes,* Prentice-Hall: Englewood Cliffs, New Jersey.

Hall, Richard H. and Engels, Gloria V. (1974) “Autonomy and Expertise: Threats and Barriers to Occupational Control” in *Varieties of Work Experience*, Stewart, Phillis and Cantor, Muriel eds. Pgs. 325-333. Wiley and Sons: New York.

Hall, Richard H., et. al. (1967) “Organizational Size, Complexity, and Formalization,” *American Sociological Review*, 32(6):903-12.

Hammel, Michael J. (2002) “Linux Goes to the Movies” *Salon*, November 1st. Available at: http://archive.salon.com/tech/feature/2001/11/01/linux_hollywood/print.html

Hebden, J. E. (1975) "Patterns of Work Identification" *Sociology of Work and Occupations*. 2, 2: 107-132.

Heckscher, Charles (1994) “Defining the Post-Bureaucratic Type” In *The Post-Bureaucratic Organization: New Perspectives on Organizational Change*, pgs. 14-62, Charles Heckscher and Anne Donnellon, eds. Sage Publications: Thousand Oaks, CA.

Henson, Val (2002) “How to Encourage Women in Linux” Online at: http://www.tldp.org/HOWTO/Encourage-Women-Linux-HOWTO/index.html

Heydebrand, Wolf V. (1989) "New Organizational Forms" *Work and Occupations*. 16, 3:323-57.

Himanen, Pekka (2001) *The Hacker Ethic and the Spirit of the Information Age* Random House: New York, NY.

Hyman, Risa (1993) “Creative Chaos in High-Performance Teams: an Experience Report,” *Communications of the ACM,* 36 (10): 56-60.

Joinson, Carla (1999) “Teams at Work” *HR Magazine*. v.44 no.5 (May 1999): 30-6.

Kidder, Tracy (1981) *The Soul of a New Machine*. Little, Brown and Company: Boston, MA.

Knuth, Donald E. (1986) [1976] "Computer Programming as an Art" in *ACM Turing Award Lectures: The First Twenty Years*. Pgs. 33-46. ACM Press: New York.

Kornhauser, William (1962) *Scientists in Industry: Conflict and Accommodation*. Berkeley UP.

Kraft, Philip. (1977) *Programmers and Managers: The Routinization of Computer Programming in the United States*. Springer-Verlag: New York.

Kraft, Philip (1979) “The Routinizing of Computer Programming” *Sociology of Work and Occupations*, 6-2: 139-55.

Kraut, Robert E. and Street, Lynn A. (1995) "Coordination in Software Development" *Communications of the ACM.* 38, 3: 69-81.

Kunda, Gideon. (1993*) Engineering Culture: Control and Commitment in a High-Tech Organization.* Temple University Press.

Lakhani, Karim R. et. al. (2002) "The Boston Consulting Group/OSDN Hacker Survey," Available online at http://www.osdn.com/bcg/

Larsson, Mats. (1998*) The Transparent Market : Management Challenges in the Electronic Age.* St. Martin's Press: New York.

Lawler, Edward (1996) *From the Ground Up: Six Principles for Building the New Logic Corporation.* Joey-Bass: New York.

Lawler, Edward, et. al. (1995) *Creating High Performance Organizations: Practices and Results of Employee Involvement and Total Quality Management in Fortune 1000 Companies.* Jossey Bass: San Francisco, CA.

Lawrence, Thomas (1998) "Examining Resources in an Occupational Community: Reputation in Canadian Forensic Accounting," *Human Relations*, 51(9):1103-31.

Lerner, Josh and Jean Tirole (2001) "The Open Source Movement: Key Research Questions" *European Economic Review Papers and Proceedings*, 35: 819-826.

Levy, Stephen (1984) *Hackers: Heroes of the Computer Revolution*. Doubleday: New York.

Ljungberg, Jan (2000) "Open Source Movements as a Model for Organizing" *European Journal of Information Systems*, 9 (4): 208-216

Loeske, D.R. and Songuist, J.A. (1979) "The Computer Worker in the Labor Force," *Sociology of Work and Occupations*, 6: 156-183

Lohr, Steve (2002) "Balancing Linux and Microsoft" *New York Times*. 9 Sept. 2002: Business.

Madanmohan, T.R. & Siddhesh Navelkar (2002) "Roles and Knowledge Management in Online Technology Communities: An Ethnography Study." Online at: http://opensource.mit.edu/papers/madanmohan2.pdf

Margolis, Jane and Fisher, Allan (2001) *Unlocking the Clubhouse: Women in Computing*. MIT University Press.

Markham, Annette N (1998) *Life Online: Researching Real Experience in Virtual Space*. Altamira Press: Walnut Creek, CA.

Markoff, John (2002) “Microsoft and Free Software at the Same Show? It’s True” *New York Times*, August 10th: Technology: 5.

Martin, Joanne, et. al. (1998) “An Alternative to Bureaucratic Impersonality and Emotional Labor: Bounded Emotionality at The Body Shop,” *Administrative Science Quarterly*, 43:429-69.

McMahon, Steve (2003) “ROI, Security Driving IT Employment Trends,” *Datamation*, June 13, 2003. Online at: http://itmanagement.earthweb.com/career/article.php/2221691

Meiksins, Peter and Smith, Chris (1993) "Organizing Engineering Work: A Comparative Analysis" *Work and Occupations*. 20, 2: 123-46.

Meyers, Colin et. al. (1997) *The Responsible Software Engineer: Selected Readings in IT Professionalism*. Springer: London, UK.

Mills, C.W. (1956) *White Collar*, Oxford UP: New York.

Mills, Donald L. and Vollmer, Howard M. (1966) *Professionalization*. Prentice-Hall: Englewood Cliffs, NJ.

Mockus, Audris et. al. (2000) “A Case Study of Open Source Software Development: The Apache Sever” *ACM Proceedings of ICSE 2000*.

Morrill, Calvin (1991) "Conflict Management, Honor, and Organizational Change" *American Journal of Sociology*. 97, 3: 585-621.

Nidumolu, Sarma (1995) "The Effect of Coordination and Uncertainty on Software Project Performance" *Information Systems Research* 6, 3: 191-219.

Osterman, Paul (1994) "How Common Is Workplace Transformation and Who Adopts It?" *Industrial and Labor Relations Review*. 47, 2: 173-88.

Ouchi, William G. (1981) *Theory Z: How American Business can Meet the Japanese Challenge.* Addison-Wesley: Reading, MA.

Ouchi, William G. (1980) "Markets, Bureaucracies, and Clans*" Administrative Science Quarterly*. 25: 129-141.

Pandit, Naresh (1996) “The Creation of Theory: A Recent Application of the Grounded Theory Method” *The Qualitative Report*, 2 (2). Also online at: http://www.nova.edu/ssss/QR/QR2-4/pandit.html

Parnas, David (1997) “Software Engineering: An Unconsummated Marriage,” *Communications of the ACM*, 40 (9): 128.

Perkin, Harold (1996) *The Third Revolution: Professional Elites in the Modern World*. Routledge: New York.

Perrucci, Robert and Gerstl, Joel E. (1969) *Profession Without Community: Engineers in American Society.* Random House: New York.

Pettigrew, Andrew M. (1973a) *The Politics of Organizational Decision-making*. Tavistock Publications: London, UK.

Pettigrew, Andrew M. (1973b) "Occupational Specialization as an Emergent Process" *Sociological Review*. 21, 2: 255-78,

Powell, Walter W. (1998) "Learning from Collaboration: Knowledge and Networks in the Biotechnology and Pharmaceutical Industries" *California Management Review*. 40, 3: 228-240.

Raymond, Eric (2002) "The Cathedral and the Bazaar" available at: http://www.tuxedo.org/~esr/writings/cathedral-bazaar/

Ruhleder, Karen (2000) "The Virtual Ethnographer: Fieldwork in Distributed Electronic Environments" in *Field Methods*, v12, no. 1, Feb. 2000: 3-17.

Rutter, Jason and Smith, Greg (2002) "Ethnographic Presence in Nebulous Settings: A Case Study," Paper presented to the ESRC Virtual Methods seminar series. CRICT, Brunel University, April 19th, 2002. Also available online at: http://www.cric.ac.uk/cric/Jason_Rutter/papers/brunel.pdf

Schellenberg, Kathryn (1996) "Taking It or Leaving It: Instability and Turnover in a High-Tech Firm" *Work and Occupations* 23, 2: 190-213.

Smith, Vicki (1997) "New Forms of Work Organization," *Annual Review of Sociology*, 23: 315-39.

Stalder and Hirsch (2001) "Open Source Intelligence" *First Monday: Peer Reviewed Journal on the Internet*, http://www.firstmonday.dk/issues/issue7_6/stalder/

Stinchcombe, Arthur (1953) "Bureaucratic and Craft Administration of Production: A Comparative Study" *American Journal of Sociology*. 58: 371-880.

Strauss, Anselm et. al. (1964) *Psychiatric Ideologies and Institutions*. The Free Press of Glencoe: London.

Tampoe, M. (1993) "Motivating Knowledge Workers: The Challenge for the 1990s," *Long Range Planning*, 26(3).

Taplin, Ian M. (2001) "Managerial Resistance to High Performance Workplace Practices" *Research in the Sociology of Work*. 10: 1-24.

Thompson, Nicholas (2000) “Reboot! How Linux and Open-Source Development Could Change the Way We Get Things Done” *Washington Monthly*, March. Available at: http://www.washingtonmonthly.com/features/2000/0003.thompson.html

Touraine, Alain (1974), *The Post-industrial Society: Tomorrow's Social History -Classes, Conflicts and Culture in the Programmed Society,* trans. by Leonard F.X. Mayhew, Wildwood House: London.

Trice, Harrison M. (1993) *Occupational Subcultures in the Workplace*. ILR Press: New York.

Van de Venn, Andrew, et. al. (1976) “Determinants of Coordination Modes within Organizations,” *American Sociological Review,* 41: 322-38.

Van Maanen, John and Barley, Stephen R. (1984) "Occupational Communities: Culture and Control in Organizations" in *Research in Organizational Behavior*, vol. 6: 287-365.

Vann, Kathryn L. (2001) *The Duplicity of Practice*. Ph.D. Dissertation. Regents of the University of California. University of California, San Diego. Co-Chairs: Susan Leigh Star, Department of Communication and Science Studies Program; Michael Cole, Department of Communication and Human Development Program.

Wang, Huaiquing and Wang, Chen (2001) “Open Source Software Adoption: A Status Report” *IEEE Software*. March/April 2001, 90-95.

Weber, Max.(1922) "Bureaucracy." In H. H. Gerth and C. Wright Mills.(1946) *Max Weber: Essays in Sociology*. Oxford University Press.

Weinberg, Gerald M. (1971) *The Psychology of Computer Programming*. Van Nostrand, Reinhold: New York.

Weiss, Robert S. (1994*) Learning Form Strangers: The Art and Method of Qualitative Interview Studies.* Free Press: New York.

Wenger, Etienne (1998) *Communities of Practice : Learning, Meaning, and Identity.* Cambridge UP: New York, NY.

Whalley, Peter (1991) “Negotiating the Boundaries of Engineering: Professionals, Managers and Manual Work,” *Research in the Sociology of Organizations,* 8: 191-215.

White, Terry (2001) *Reinventing the IT Department,* Butterworth-Heinemann: Oxford, UK.

Willams, Michele (2001) “In Whom We Trust: Group Membership as an Affective Context for Trust Development,” *Academy of Management Review*, 26 (3): 377-96.

Williamson, Oliver E. (1981) "The Economics of Organization: The Transaction Cost Approach" *American Journal of Sociology*. 548-577.

Wilson, Francis (1999) "Cultural Control Within the Virtual Organization” *Sociological Review*, 672-694.

Wootton, Barbara (1997) “Gender Differences in Occupational Employment” Bureau of Labor Statistics Report. Also online at: http://www.bls.gov/opub/mlr/1997/04/art2full.pdf

Young, R. and Mould, K. (1994) *Managing Information Systems Professionals.* Butterworth and Heinemann: Oxford, UK.

Zabusky, Sharon and Barley, Stephen R. (1996) "Redefining Success: Ethnographic Observations on the Careers of Technicians." Pp. 185-214 in Paul Osterman (ed.) *Broken Ladders: Managerial Careers in Transition*. Oxford, Eng: Oxford UP.

Zachary, G. Pascal (1994) *Showstopper: The Breakneck Race to Create Windows NT and the Next Generation at Microsoft*. The Free Press: New York.

Zetka, James R., Jr. (2001) "Occupational Divisions of Labor and Their Technology Politics: The Case of Surgical Scopes and Gastrointestinal Medicine" *Social Forces*. 79, 4 :1495-1520.